HCL Informix 14.10 - Enterprise Replication Guide

Contents

Chapter 1. Enterprise Replication

These topics describe the concepts of data replication using HCL Informix® Enterprise Replication, including how to design your replication system, as well as administer and manage data replication throughout your enterprise.

These topics are for database server administrators with 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 database server administration, operating- system administration, and network administration

These topics are taken from HCL Informix® Enterprise Replication *Guide*.

About Enterprise Replication

HCL Informix® Enterprise Replication generates and manages multiple copies of data at one or more sites, which allows an enterprise to share corporate data throughout its organization.

These topics provide an overview of HCL Informix® Enterprise Replication and how to administer it.

HCL Informix® Enterprise Replication technical overview

HCL Informix® Enterprise Replication is an asynchronous, log-based tool for replicating data between HCL Informix® database servers. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

At each target server, Enterprise Replication receives and applies each transaction contained in the replication data to the appropriate databases and tables as a normal, logged transaction.

Enterprise Replication Terminology

You must understand Enterprise Replication terminology.

The following terms define the data in an Enterprise Replication system and how it is treated:

- Enterprise Replication server
- Shard server
- Replication key
- Replicate
- Master Replicate
- Shadow Replicate
- Participant
- Replicate Set
- Template
- Global Catalog
- Grid

Enterprise Replication server

An Enterprise Replication server, or *replication server*, is the HCL® Informix® database server that participates in data replication.

The replication server maintains information about the replication environment, which columns are replicated, and the conditions under which the data is replicated. This information is stored in a database, **syscdr**, that the database server creates when it is initialized. Multiple database servers can be on the same physical computer, and each database server can participate in Enterprise Replication.

Shard server

A *shard server* is a database server that participates in data replication and receives horizontally partitioned (sharded) data. A *shard cluster* is the group of database servers over which a table or collection is partitioned.

Replication key

A *replication key* consists of one or more columns that uniquely identifies each replicated row. The replication key must be the same on all servers that participate in the replicate. Typically, the replication key is a primary key constraint. Otherwise, you can specify ERKEY shadow columns or another unique index as the replication key.

The replication key for a shard cluster consists of a single column, and is called a *shard key*.

Replicate

A *replicate* defines the replication *participants* and various attributes of how to replicate the data, such as frequency and how to handle any conflicts during replication.

For more information, see Define a replicate on page 100 and cdr define replicate on page 342.

Master replicate

A *master replicate* is a replicate that guarantees data integrity by verifying that replicated tables on different servers have consistent column attributes. Master replicates also support alter operations on replicated tables.

Shadow replicate

A *shadow replicate* is a copy of an existing (primary) replicate. Shadow replicates allow Enterprise Replication to manage alter and repair operations on replicated tables.

Participant

A *participant* specifies the data (database, table, and columns) to replicate and the database servers to which the data replicates.

Replicate set

A *replicate set* combines several replicates to form a set that can be administered together as a unit.

Template

A *template* provides a mechanism to set up and deploy replication for a group of tables on one or more servers. A template is especially useful if you have many tables to replicate between many servers. A template defines a group of master replicates and a replicate set for a specified group of tables that are based on attributes such as database, tables, columns, and primary keys from the master node.

You create a template by running the cdr define template command and then instantiate, or realize, it on servers with the cdr realize template command.

Global catalog

Each database server that participates in Enterprise Replication maintains tables in the **syscdr** database to track Enterprise Replication configuration information and state. For all root and nonroot replication servers, this catalog is a *global catalog* that maintains a global inventory of Enterprise Replication configuration information. The global catalog is created when you define the server for replication.

The global catalog includes the following information:

- Enterprise Replication server definitions and state
- Routing and connectivity information
- Replicate definitions and state
- Participant definitions and state
- Replicate set definitions and state
- Conflict detection and resolution rules and any associated SPL routines

Grid

A grid is a set of replication servers that you can administer as a unit. When you run SQL data definition statements from within a grid context on one server in the grid, they are propagated to all other servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can propagate external files to other servers in the grid. You can run grid queries to consolidate data from multiple grid servers.

Related information

Connect Option on page 256

Customizing the Replication Server Definition on page 99

Connect to another replication server on page 164

Asynchronous Data Replication

Enterprise Replication uses *asynchronous* data replication to update the databases that reside at a replicated site after the primary database has committed a change.

With asynchronous replication, the delay to update the replicated-site databases can vary depending on the business application and user requirements. However, the data eventually synchronizes to the same value at all sites. The major benefit of this type of data replication is that if a particular database server fails, the replication process can continue and all transactions in the replication system will be committed.

In contrast to this, *synchronous* data replication replicates data immediately when the source data is updated. Synchronous data replication uses the *two-phase commit* technology to protect data integrity. In a two-phase commit, a transaction is applied only if *all* interconnected distributed sites agree to accept the transaction. Synchronous data replication is appropriate for applications that require immediate data synchronization. However, synchronous data replication requires that all hardware components and networks in the replication system be available at all times. For more information about synchronous replication, refer to the discussion of two-phase commit in your *HCL® Informix® Administrator's Guide*.

Asynchronous replication is often preferred because it allows for system and network failures.

Asynchronous replication allows the following replication models:

- Primary-target (Primary-Target Replication System on page 35)

  All database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.
- Update-anywhere (Update-Anywhere Replication System on page 39)

  All databases have read and write capabilities. Updates are applied at all databases.

The update-anywhere model provides the greater challenge in asynchronous replication. For example, if a replication system contains three replication sites that all have read and write capabilities, conflicts occur when the sites try to update the same data at the same time. Conflicts must be detected and resolved so that the data elements eventually have the same value at every site. For more information, see Conflict Resolution on page 40.

Log-Based Data Capture

Enterprise Replication uses *log-based data capture* to gather data for replication. Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication and then evaluates the row images.

Log-based data capture takes changes from the logical log and does not compete with transactions for access to production tables. Log-based data-capture systems operate as part of the normal database-logging process and thus add minimal overhead to the system.

Two other methods of data capture, which Enterprise Replication does not support, include:

- Trigger-based data capture

  A trigger is code in the database that is associated with a piece of data. When the data changes, the trigger activates the replication process.

- Trigger-based transaction capture

  A trigger is associated with a table. Data changes are grouped into transactions and a single transaction might trigger several replications if it modifies several tables. The trigger receives the whole transaction, but the procedure that captures the data runs as a part of the original transaction, thus slowing down the original transaction.

High Performance

Enterprise Replication provides high performance by not overly burdening the data source and by using networks and all other resources efficiently.

Because Enterprise Replication captures changes from the logical log instead of competing with transactions that access production tables, Enterprise Replication minimizes the effect on transaction performance. Because the capture mechanism is internal to the database, the database server implements this capture mechanism efficiently. For more information, see Log-Based Data Capture on page 6.

All Enterprise Replication operations are performed in parallel, which further extends the performance of Enterprise Replication.

High Availability

Because Enterprise Replication implements asynchronous data replication, network and target database server outages are tolerated. In the event of a database server or network failure, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available.

If high availability is critical, you can use high-availability clusters in conjunction with Enterprise Replication. High-availability clusters support synchronous data replication between database servers: a primary server, which can participate in Enterprise Replication, and one or more secondary servers, which do not participate in Enterprise Replication. If a primary server in a high-availability cluster fails, a secondary server can take over the role of the primary server, allowing it to participate in Enterprise Replication. Client connections to the original primary server can be automatically switched to the new standard server.

For more information on using high-availability clusters with Enterprise Replication, see Using High-Availability Clusters with Enterprise Replication on page 87.

Consistent Information Delivery

HCL Informix® Enterprise Replication protects data integrity. All HCL Informix® Enterprise Replication transactions are stored in a reliable queue to maintain the consistency of transactions.

HCL Informix® Enterprise Replication uses a data-synchronization process to ensure that transactions are applied at the target database servers in any order equivalent to the order that they were committed on the source database server. If Enterprise Replication can preserve the consistency of the database, Enterprise Replication might commit transactions in a slightly different order on the target database.

If update conflicts occur, HCL Informix® Enterprise Replication provides built-in automatic conflict detection and resolution. You can configure the way conflict resolution behaves to meet the needs of your enterprise. For more information, see Conflict Resolution on page 40.

Repair and Initial Data Synchronization

Enterprise Replication provides initial data synchronization and multiple methods to repair replicated data.

You can easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate, by specifying an initial synchronization. Initial synchronization can be run online while replication is active.

If replication has failed for some reason, you can repair replicated data by running the cdr sync replicate or cdr sync replicateset command to resynchronize data and correct data mismatches between replicated tables. You can repair data while replication is active.

You can also repair data after replication has failed by using ATS and RIS files.Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied.

Related information

Initially Synchronizing Data Among Database Servers on page 120

Repairing Failed Transactions with ATS and RIS Files on page 187

Resynchronizing Data among Replication Servers on page 177

Flexible Architecture

Enterprise Replication allows replications based on specific business and application requirements and does not impose model or methodology restrictions on the enterprise.

Enterprise Replication supports both primary-target and update-anywhere replication models.

Enterprise Replication supports the following network topologies:

- Fully connected

  Continuous connectivity between all participating database servers.
- Hierarchical tree

  A parent-child configuration that supports continuous and intermittent connectivity.
- Forest of trees

  Multiple hierarchical trees that connect at the root database servers.

You can add High-Availability Data Replication to any of these topologies.

Enterprise Replication supports all built-in HCL® Informix® data types, as well as extended and user-defined data types.

Enterprise Replication operates in LAN, WAN, and combined LAN/WAN configurations across a range of network transport protocols.

Enterprise Replication supports the Global Language Support (GLS) feature, which allows HCL® Informix® products to handle different languages, regional conventions, and code sets.

Related information

Primary-Target Replication System on page 35

Update-Anywhere Replication System on page 39

Choosing a Replication Network Topology on page 51

Replication and data types on page 29

Global language support for replication on page 22

Centralized Administration

Enterprise Replication allows administrators to easily manage all the distributed components of the replication system from a single point of control.

You can use the command-line utility (CLU) to administer the replication system from your system command prompt and connect to other servers involved in replication, as necessary. For information, see The cdr utility on page 253.

Ease of Implementation

Enterprise Replication provides templates to allow easy set up and deployment of replication for clients with large numbers of tables to replicate. Administrators of Enterprise Replication can use templates to develop scripts and with only a few commands can set up replication over a large number of server nodes. Without using templates, many individual commands must be run. Using templates, you can also easily add a new server into your replication environment and optionally create and populate new database tables.

First, you create a template using the **cdr define template** command. This defines the database, tables, and columns and the characteristics of the replicates that will be created. You can view information about a template by using the **cdr list template** command from a non-leaf node.

Second, you instantiate the template on the servers where you want to replicate this data by running the **cdr realize template** command. If the table already exists on a node, Enterprise Replication verifies it matches the template definition. If the table does not exist on a node, Enterprise Replication can optionally create the table. Enterprise Replication can also optionally perform an initial data synchronization on all servers where you realize the template.

You can delete templates that you no longer need using the **cdr delete template** command.

See Set up replication through templates on page 121 for more information. All replication commands mentioned in this section are described in detail in The cdr utility on page 253.

Network Encryption

Enterprise Replication supports the same network encryption options that you can use with communications between server and clients to provide complete data encryption.

You can use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures privacy and integrity of data transmitted over the network, for connections between Enterprise Replication servers. For information on using the SSL protocol, see Secure sockets layer protocol on page .

You can use encryption configuration parameters to provide data encryption with a standard cryptography library. A message authentication code (MAC) is transmitted as part of the encrypted data transmission to ensure data integrity. This is the same type of encryption provided by the ENCCSM communications support module for non-replication communication. Enterprise Replication shares the same ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE, and ENCRYPT_SWITCH configuration parameters with high availability clusters. Enterprise Replication encryption configuration parameters are documented in Enterprise Replication configuration parameter and environment variable reference on page 498.

Enterprise Replication cannot accept a connection that is configured with a communications support module. To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server, one with CSM and one without. For more information, see Configuring network encryption for replication servers on page 61.

Note: Support for Communication Support Module (CSM) is removed starting Informix Server 14.10.xC9 . You should use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) instead.

How Enterprise Replication Replicates Data

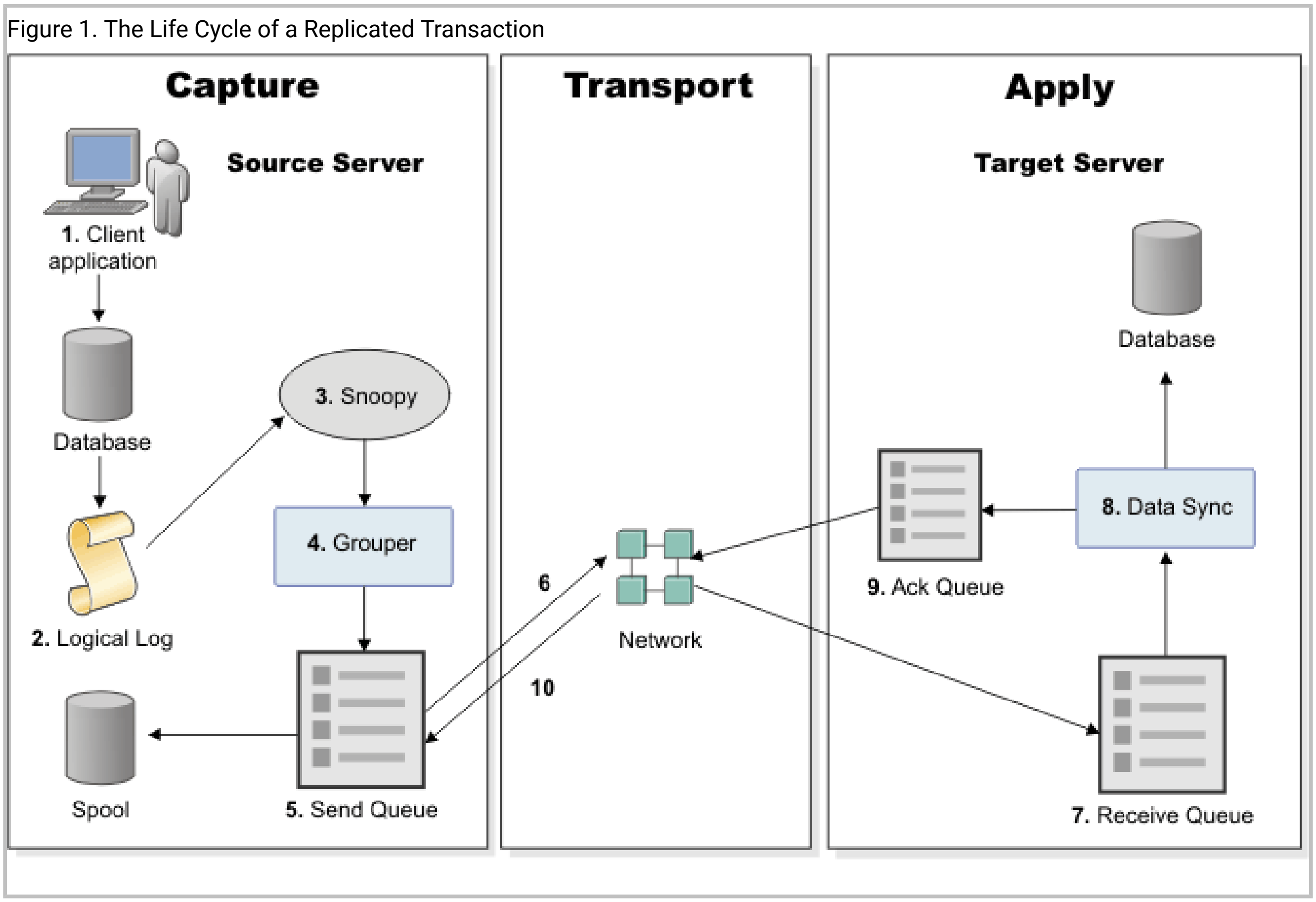
Before you can replicate data, you must declare a database server for replication and define the *replicates* (the data to replicate and the database servers that participate in replication). To declare a database server for replication, see Defining Replication Servers on page 93. To define replicates, see Define a replicate on page 100. Replication Examples on page 596, has simple examples of declaring replication servers and defining replicates.

After you define the servers and replicates, Enterprise Replication replicates data in three phases:

1. Data Capture on page 12
2. Data Transport on page 17
3. Applying replicated data on page 17

The following diagram shows these three phases of replication and the Enterprise Replication components that perform each task.

Figure 1. The Life Cycle of a Replicated Transaction



As shown in the diagram, the following process describes how Enterprise Replication replicates a transaction:

1. A client application performs a transaction in a database that is defined as a replicate.
2. The transaction is put into the logical log.
3. The log capture component, also known as the snoopy component, reads the logical log and passes the log records onto the grouper component.
4. The grouper component evaluates the log records for replication and groups them into a message that describe the operations that were in the original transaction.
5. The grouper component places the message in the send queue. Under certain situations, the send queue spools messages to disk for temporary storage.
6. The send queue transports the replication message across the Enterprise Replication network to the target server.
7. The replication message is placed in the receive queue at the target server.
8. The data sync component applies the transaction in the target database. If necessary, the data sync component performs conflict resolution.
9. An acknowledgment that the message was successfully applied is placed in the acknowledgment queue.
10. The acknowledgment message is sent back to the source server.

Data Capture

As the database server writes rows to the logical log, it marks rows that should be replicated. Later, Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication.

HCL® Informix® database servers manage the logical log in a circular fashion; the most recent logical-log entries write over the oldest entries. Enterprise Replication must read the logical log quickly enough to prevent new logical-log entries from overwriting the logs Enterprise Replication has not yet processed.

If the database server comes close to overwriting a logical log that Enterprise Replication has not yet processed, by default, user transactions are blocked until Enterprise Replication advances. You can specify other responses to the potential for overwriting the Enterprise Replication replay position.

The row images that participate in replication are passed to Enterprise Replication for further evaluation.

Row Images

Enterprise Replication evaluates the initial and final images of a row and any changes that occur between the two row images to determine which rows to replicate. Each row image contains the data in the row and the action that is performed on that row.

A row might change more than once in a particular transaction. For example, a transaction might insert and then update a row before committing. Enterprise Replication evaluates the net effect (final state) of a transaction based on the row buffers in the log. Enterprise Replication then determines what must be replicated, based on the net effect, the initial state of the row, and whether the replicate definition (in particular, the WHERE clause) applies to the initial and final state. Enterprise Replication evaluates the row-image type (INSERT, UPDATE, DELETE), the results of evaluating the replicate WHERE clause for both the initial and final image, and whether the replication key changes as a result of the transaction.

The following table shows the logic that determines which rows are candidates for replication. The source and destination tables are assumed to be initially synchronized (identical before replication begins). If the tables were not synchronized, anomalous behavior might result.

Table 1. Enterprise Replication Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Replication-Key Changed?	Send to Destination Database Server	Comments
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
INSERT	T *or* F	UPDATE	T	Yes *or* no	INSERT with final row image	Inserts final data of transaction
INSERT	T *or* F	UPDATE	F	Yes *or* no	Nothing	Final evaluation determines no replication

Table 1. Enterprise Replication Evaluation Logic (continued)

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Replication-Key Changed?	Send to Destination Database Server	Comments
UPDATE	T	DELETE	T *or* F	Yes *or* no	DELETE with initial row image	Net result of transaction is delete
UPDATE	F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
UPDATE	T	UPDATE	T	Yes	DELETE with initial row image and INSERT with final row image	Ensures old replication key does not replicate
UPDATE	T	UPDATE	T	No	UPDATE with final row image	Simple update
UPDATE	T	UPDATE	F	Yes *or* no	DELETE with initial row image	Row no longer matches replicate definition
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image	Row now matches replicate definition
UPDATE	F	UPDATE	F	Yes *or* no	Nothing	No match exists, and therefore, no replication

The following rules apply to the information in the table:

- The initial image is the before image of the transaction in the logical log.
- The replicate evaluates to T (true) or F (false).
- The final image is the image of the transaction that is replicated.

After Enterprise Replication identifies transactions that qualify for replication, Enterprise Replication transfers the transaction data to a queue.

Evaluate rows for updates

Enterprise Replication evaluates rows for replication-key updates, for WHERE-clause column updates, and for multiple updates to the same row.

The following list describes an occurrence in a transaction and the Enterprise Replication action:

- Replication-key updates

  Enterprise Replication translates an update of the replication key into a delete of the original rows and an insert of the row images with the new replication key. If triggers are enabled on the target system, insert triggers are run.

- WHERE-clause column updates

  If a replicate includes a WHERE clause in its data selection, the WHERE clause imposes selection criteria for rows in the replicated table.

  - If an update changes a row so that it no longer passes the selection criteria on the source, it is deleted from the target table. Enterprise Replication translates the update into a delete and sends it to the target.
  - If an update changes a row so that it passes the selection criteria on the source, it is inserted into the target table. Enterprise Replication translates the update into an insert and sends it to the target.
- Multiple-row images in a transaction

  Enterprise Replication compresses multiple-row images and only sends the net change to the target. Because of this, triggers might not execute on the target database. For more information, see Triggers on page 26.

Enterprise Replication supports the replication of BYTE and TEXT data types (simple large objects) and BLOB and CLOB data types (smart large objects), and opaque user-defined data types, as well as all built-in HCL® Informix® data types. However, Enterprise Replication implements the replication of these types of data somewhat differently from the replication of other data types. For more information, see Replication of large objects on page 31, and Replication of opaque user-defined data types on page 33.

Send queues and receive queues

Enterprise Replication uses send and receive queues to receive and deliver replication data to and from database servers that participate in a replicate.

Send queue

Enterprise Replication stores replication data in memory to be delivered to target database servers that participate in a replicate. If the send queue fills, Enterprise Replication spools the send-queue transaction records to a dbspace and the send-queue row data to an sbspace.

Receive queue

Enterprise Replication stores replication data in memory at the target database server until the target database server acknowledges receipt of the data. If the receive queue fills as a result of a large transaction, Enterprise Replication spools the receive queue transaction header and replicate records to a dbspace and the receive queue row data to an sbspace.

The data contains the filtered log records for a single transaction. Enterprise Replication stores the replication data in a stable (recoverable) send queue on the source database server. Target sites acknowledge receipt of data when it is applied to or rejected from the target database.

If a target database server is unreachable, the replication data remains in a stable queue at the source database server. Temporary failures are common, and no immediate action is taken by the source database server; it continues to queue transactions. When the target database server becomes available again, queued transactions are transmitted and applied.

If the target database server is unavailable for an extended period, the send queue on the source database server might use excessive resources. In this situation, you might not want to save all transactions for the target database server. To prevent unlimited transaction accumulation, you can remove the unavailable target database server from the replicate. Before the database server that is removed rejoins any replicate, however, you must synchronize (bring tables to consistency) with the other database servers.

Related information

Transaction processing impact on page 20

Setting Up Send and Receive Queue Spool Areas on page 66

Preventing Memory Queues from Overflowing on page 212

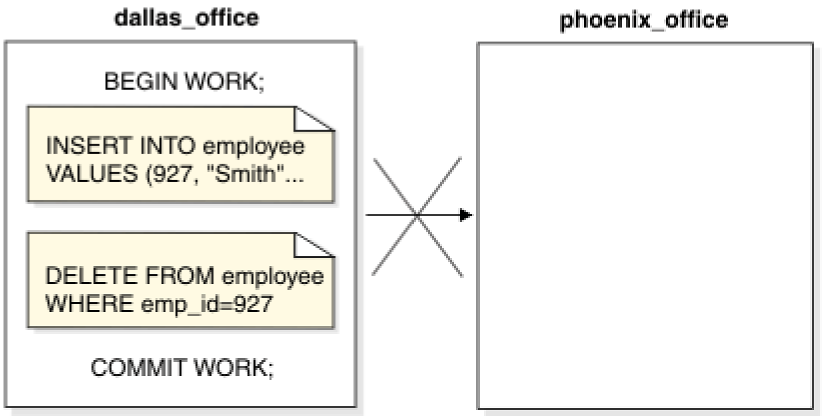
Applying replicated data on page 17

Data Evaluation Examples

Figure 2: Insert Followed by a Delete on page 15, Figure 3: Insert Followed by an Update on page 16, and Figure 4: Update; Not Selected to Selected on page 16 show three examples of how Enterprise Replication uses logic to evaluate transactions for potential replication.

Figure 2. Insert Followed by a Delete

Replicate SQL=SELECT emp_id, salary FROM employee WHERE exempt = "N";



```
BEGIN WORK;
INSERT INTO employee
VALUES (927, "Smith"...
DELETE FROM employee
WHERE emp_id=927
COMMIT WORK;
```

Figure 2: Insert Followed by a Delete on page 15 shows a transaction that takes place at the Dallas office. Enterprise Replication uses the logic in Table 2: Insert Followed by a Delete Evaluation Logic on page 15 to evaluate whether any information is sent to the destination database server at the Phoenix office.

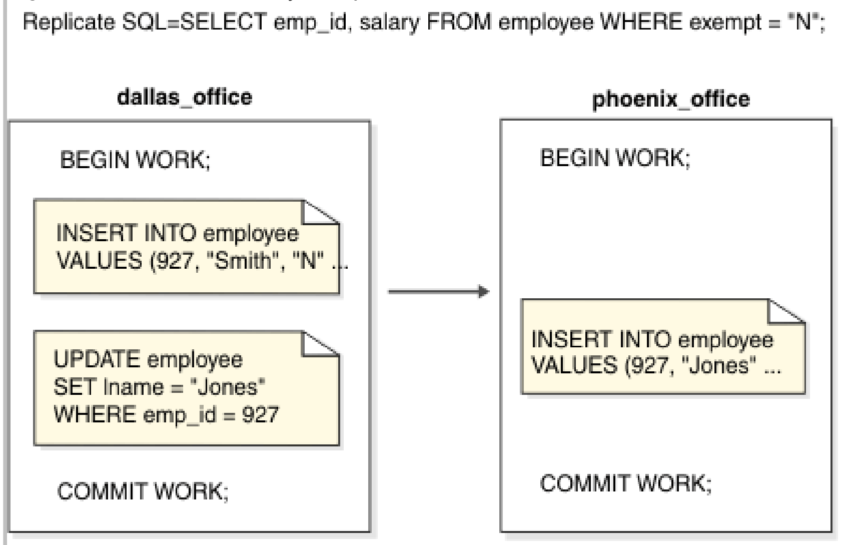
Table 2. Insert Followed by a Delete Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing

Enterprise Replication determines that the insert followed by a delete results in no replication operation; therefore, no replication data is sent.

In Figure 3: Insert Followed by an Update on page 16, Enterprise Replication uses the logic in Table 3: Insert Followed by An Update Evaluation Logic on page 16 to evaluate whether any information is sent to the destination database server.

Figure 3. Insert Followed by an Update



```
Replicate SQL=SELECT emp_id, salary FROM employee WHERE exempt = "N";
```

dallas_office:

```
BEGIN WORK;
INSERT INTO employee
VALUES (927, "Smith", "N" ...
UPDATE employee
SET lname = "Jones"
WHERE emp_id = 927
COMMIT WORK;
```

phoenix_office:

```
BEGIN WORK;
INSERT INTO employee
VALUES (927, "Jones" ...
COMMIT WORK;
```

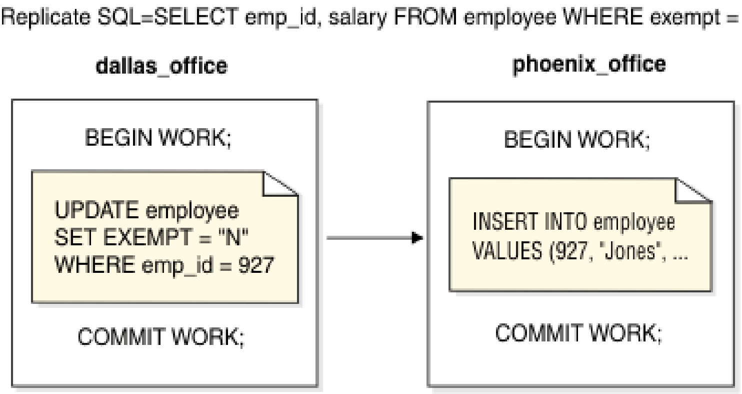
Table 3. Insert Followed by An Update Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	UPDATE	T	Yes or no	INSERT with final row image

The replicate WHERE clause imposes the restriction that only rows are replicated where the exempt column contains a value of "N." Enterprise Replication evaluates the transaction (an insert followed by an update) and converts it to an insert to propagate the updated (final) image.

In Figure 4: Update; Not Selected to Selected on page 16, Enterprise Replication uses the logic in Table 4: Update; Not Selected to Selected Evaluation Logic on page 17 to evaluate whether any information is sent to the destination database server.

Figure 4. Update; Not Selected to Selected



```
Replicate SQL=SELECT emp_id, salary FROM employee WHERE exempt = "N";
```

dallas_office:

```
BEGIN WORK;
UPDATE employee
SET EXEMPT = "N"
WHERE emp_id = 927
COMMIT WORK;
```

phoenix_office:

```
BEGIN WORK;
INSERT INTO employee
VALUES (927, "Jones", ...
COMMIT WORK;
```

Table 4. Update; Not Selected to Selected Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image

The example shows a replicate WHERE clause column update. A row that does not meet the WHERE clause selection criteria is updated to meet the criteria. Enterprise Replication replicates the updated row image and converts the update to an insert.

Data Transport

Enterprise Replication ensures that all data reaches the appropriate server, regardless of a network or system failure. In the event of a failure, Enterprise Replication stores data until the network or system is operational. Enterprise Replication replicates data efficiently with a minimum of data copying and sending.

Applying replicated data

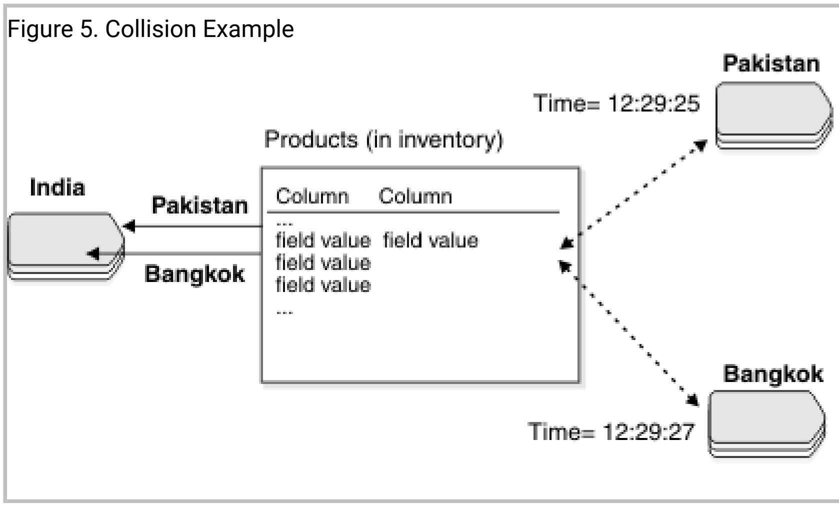
HCL Informix® Enterprise Replication uses a data-synchronization process to apply the replicated data to target database servers.

The target database servers acknowledge receipt of data when the data is applied to the target database. Data modifications that results from synchronization, including modifications that result from trigger invocation, are not replicated. The data-synchronization process ensures that transactions are applied at the target database servers in an order equivalent to the order that they were committed on the source database server. If consistency can be preserved, Enterprise Replication might commit transactions out of order on the target database.

When Enterprise Replication applies replication data, it checks to make sure that no *collisions* exist. A collision occurs when two database servers update the same data simultaneously. Enterprise Replication reviews the data one row at a time to detect a collision.

If Enterprise Replication finds a collision, it must resolve the conflict before applying the replication data to the target database server.

Figure 5. Collision Example



The previous illustration shows a situation that yields a conflict. Pakistan updates the row two seconds before Bangkok updates the same row. The Bangkok update arrives at the India site first, and the Pakistan update follows. The Pakistan time is earlier than the Bangkok time. Because both updates involve the same data and a time discrepancy exists, Enterprise Replication detects a collision.

For more information, see Conflict Resolution on page 40.

Enterprise Replication scans to see if the same replication key exists in the target table or in the associated *delete table*, or if a replication order error is detected. A delete table stores the row images of deleted rows. A replication order error is the result of replication data that arrives from different database servers with one of the following illogical results:

- A replicated DELETE that finds no row to DELETE on the target
- An UPDATE that finds no row to UPDATE on the target
- An INSERT that finds a row that exists on the target

Related information

Send queues and receive queues on page 14

Planning and designing for Enterprise Replication

Before you set up your replication system, plan how to include Enterprise Replication into your database server environment, design your database schema by following Enterprise Replication requirements, and then design your replication system between database servers.

Plan for Enterprise Replication

Before you design a replication system, you must understand how Enterprise Replication interacts with the database server and the other requirements of Enterprise Replication. Many aspects of the Informix® database server can affect how you deploy Enterprise Replication.

Enterprise Replication Server administrator

You need special privileges to run most Enterprise Replication commands.

The Enterprise Replication server administrator must have HCL® Informix® Database Server Administrator (DBSA) privileges to configure and manage Enterprise Replication or be user **informix** (UNIX™) or a member of the **Informix-Admin** group (Windows™).

To configure and manage Enterprise Replication, you must have one of the following roles or privileges:

- Be the owner of a non-root server
- Have the Database Server Administrator (DBSA) privilege
- Be user **informix** (UNIX™) or a be a member of the **Informix-Admin** group (Windows™)

All servers in the replication domain must have the same owner.

Related information

Interpret the cdr utility syntax on page 253

Defining Replication Servers on page 93

grant admin argument: Grant privileges to run SQL administration API commands on page

Asynchronous propagation conflicts

Enterprise Replication asynchronously propagates many control operations through the Enterprise Replication network. Avoid operations that might conflict during propagation.

When you perform administrative functions using Enterprise Replication, the status that returns from those operations indicates the success or failure of the operation at the database server to which you are directly connected. The operation might still be propagating through the other Enterprise Replication database servers in the network at that time. It might take a significant amount of time before the operation is propagated to database servers that are not connected to the Enterprise Replication network at all times.

Due to this asynchronous propagation, avoid performing control operations in quick succession that might directly conflict with one another without verifying that the first operation was successfully propagated through the entire enterprise network. Specifically, avoid deleting Enterprise Replication objects such as replicates, replicate sets, and Enterprise Replication servers, and immediately recreating those objects with the same name. Doing so can cause failures in the Enterprise Replication system at the time of the operation or later. These failures might manifest themselves in ways that do not directly indicate the source of the problem.

If you must recreate a deleted definition with the same name, run the cdr check queue command to make sure that the command is complete on all servers before recreating the definition.

You can also use a different name for the new object (for example, delete replicate **a.001** and recreate it as **a.002**) or wait until the delete action was successfully propagated through the entire Enterprise Replication system before you recreate the object.

If you must recreate a deleted definition, use a different name for the new object (for example, delete replicate **a.001** and recreate it as **a.002**) or wait until the delete action was successfully propagated through the entire Enterprise Replication system before you recreate the object.

Back up and restore of replication servers

You can back up and restore database servers that participate in Enterprise Replication.

Do not stop Enterprise Replication before performing a backup on database servers that participate in replication.

Warm restores are not permitted. You must perform a cold restore up to the current log of all relevant dbspaces on Enterprise Replication servers before resuming replication.

If the restore did not include all the log files from the replay position, or the system was not restored to the current log file, you must advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the cdr cleanstart command followed by the cdr sync replicate command to synchronize the server.

Compression of replicated data

You can compress and uncompress data in replicated tables to reduce the amount of needed disk space.

You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Performing these operations on one Enterprise Replication server does not affect the data on any other Enterprise Replication server.

Attention: After you uncompress data on one server, do not remove any compression dictionaries that another Enterprise Replication server needs.

Related information

Compression on page

Transaction processing impact

Many variables affect the impact that replicating data has on your transaction processing.

Replication volume

To determine replication volume, you must estimate how many data rows change per day. For example, an application issues a simple INSERT statement that inserts 100 rows. If this table is replicated, Enterprise Replication must propagate and analyze these 100 rows before applying them to the targets.

Distributed transactions

A *distributed transaction* is a transaction that commits data in a single transaction over two or more database servers.

Outside of the replication environment, Informix® uses a two-phase commit protocol to ensure that the transaction is either committed completely across all servers involved or is not committed on any server. For more information about the two-phase commit protocol, see the *HCL® Informix® Administrator's Guide*.

In a replication environment, when a distributed transaction is committed across the source servers, each part of the transaction that applies to the local server is written to the local logical logs. When Enterprise Replication retrieves the transaction from the logical logs and forms its transaction data, it is unable to identify the separate transaction data as the original single transaction.

This situation might result in Enterprise Replication applying one transaction successfully while aborting another. Another result might be a time lapse between the application of one transaction and another (depending on how much transaction data is in each server's send queue and the state of the server).

Large transactions

While Enterprise Replication is able to handle large transactions, it is optimized for small transactions. For best performance, avoid replicating large transactions.

Large transactions are handled with a grouper paging file in temporary smart large objects. Enterprise Replication can process transactions up to 4 TB in size. For more information, see Setting Up the Grouper Paging File on page 71. You can view Enterprise Replication grouper paging statistics with the onstat -g grp pager command (see onstat -g grp: Print grouper statistics on page 556).

Instead of using Enterprise Replication to perform a batch job, use BEGIN WORK WITHOUT REPLICATION to run the batch job locally on each database server. For more information, see Blocking Replication on page 76.

Related information

Send queues and receive queues on page 14

Preventing Memory Queues from Overflowing on page 212

SQL statements and replication

You can run most SQL statements while replication is active. For some statements, however, you must set alter mode or stop replication.

You can run the following SQL statements with no limitations while Enterprise Replication is active:

- ADD INDEX
- ALTER INDEX . . . TO CLUSTER
- ALTER FRAGMENT
- ALTER INDEX
- ALTER TABLE (except for the replication key)
- CREATE CLUSTER INDEX
- CREATE SYNONYM
- CREATE TRIGGER
- CREATE VIEW
- DROP INDEX
- DROP SYNONYM
- DROP TRIGGER
- DROP VIEW
- RENAME COLUMN
- RENAME DATABASE
- RENAME TABLE
- SET *object mode* (no disabling of replication key constraint)
- START VIOLATIONS TABLE

- STOP VIOLATIONS TABLE
- TRUNCATE TABLE

After you define Enterprise Replication on a table by including that table as a participant in a replicate, you cannot exclusively lock a database that is involved in replication (or perform operations that require an exclusive lock). However, you can exclusively lock a table in a database.

You can rename both dbspaces and sbspaces while HCL Informix® Enterprise Replication is active.

You cannot use the DROP TABLE SQL statement against a table that is included in a replicate.

You must first set alter mode with the cdr alter command before you can make these changes:

- Add shadow columns:
  - ALTER TABLE ... ADD CRCOLS;
  - ALTER TABLE ... ADD REPLCHECK;
  - ALTER TABLE ... ADD ERKEY
- Remove or disable the replication key constraint.
- Modify the replication key columns. For example, alter a column to add default values or other integrity constraints.
- Change the replication key from one or more columns to others. For example, if a replication key is defined on **col1**, you can change the replication key to **col2**.

You must stop replication before you make these changes:

- Drop conflict resolution shadow columns with ALTER TABLE ... DROP CRCOLS.
- Add or drop rowids.

SQL statements are limited to a maximum of 15000 bytes.

Related reference

cdr alter on page 290

Related information

Preparing Tables for a Consistency Check Index on page 78

Changing or re-creating primary key columns on page 194

Enterprise Replication shadow columns on page

Preparing Tables for Conflict Resolution on page 77

Preparing tables without primary keys on page 79

Alter, rename, or truncate operations during replication on page 188

Global language support for replication

You can replicate data in non-default locales.

An Enterprise Replication system can include databases in different locales, with the following restrictions:

- When you define a database server for Enterprise Replication, that server must be running in the U. S. English locale.

  The **syscdr** database on every Enterprise Replication server must be in the English locale.
- Replicate names can be in the locale of the database.

Code-set conversion with the GLS library requires only those code-set conversion files found in the `$INFORMIXDIR/gls/cv9` directory.

- For U.S. English, locales are handled automatically by the HCL® Informix® Client Software Development Kit (Client SDK) installation and setup.
- For non-U.S. English locales, you might need to explicitly provide the locale and conversion files.

For information about how to specify a nondefault locale and other considerations related to GLS locales, see the *HCL® Informix® GLS User's Guide*.

Related information

Flexible Architecture on page 8

Enabling code set conversion between replicates on page 108

Replication between multiple server versions

You can set up Enterprise Replication across servers of different version levels.

Enterprise Replication stores an internal version number that it communicates to other servers on initiating a connection with them. Each Enterprise Replication server instance can only use the features supported by its version level. Attempts to use features from later releases with previous versions of Enterprise Replication raise errors.

Schema design for Enterprise Replication

When you design the database and tables for replication, you must follow the requirements and restrictions for Enterprise Replication.

Unbuffered Logging

Databases on all server instances involved in replication must be created with logging.

Enterprise Replication evaluates the logical log for transactions that modify tables defined for replication. If a table defined for replication resides in a database that uses buffered logging, the transactions are not immediately written to the logical log, but are instead buffered and then written to the logical log in a block of logical records. When this occurs, HCL Informix® Enterprise Replication evaluates the buffer of logical-log records all at once. Buffered logging can require more time to flush the logs to disk. When you define a table for replication in a database created with unbuffered logging, Enterprise Replication can evaluate the transactions as they are produced.

Unlogged changes to a table, such as when data is added by a light append, can be replicated to other tables.

To create a database with unbuffered logging, use:

```
CREATE DATABASE db_name WITH LOG
```

To minimize impact on the system, HCL Informix® Enterprise Replication uses buffered logging whenever possible, even if the database is defined as unbuffered. For more information, see the section on CREATE DATABASE in the *IBM® Informix® Database Design and Implementation Guide*.

Table Types

Enterprise Replication has restrictions on the types of tables that can participate in replication.

The following table types are not supported by Enterprise Replication:

- RAW tables
- Temporary tables

  Because the database server deletes temporary tables when an application terminates or closes the database, do not include these tables in your replication environment.

Enterprise Replication imposes the following operational limitations:

- Replication is restricted to base tables. That is, you cannot define a replicate on a view or synonym. A *view* is a synthetic table, a synthesis of data that exists in real tables and other views. A *synonym* is an alternative name for a table or a view. For more information about views and synonyms, see the *IBM® Informix® Database Design and Implementation Guide*.
- Replication is not inherited by any child tables in a typed hierarchy.

For more information about table types, see *IBM® Informix® Database Design and Implementation Guide*.

Label-based access control

You cannot apply label-based access control (LBAC) to a table participating in Enterprise Replication. Nor can you define an Enterprise Replication replicate on a table that is protected by LBAC.

Out-of-Row Data

Enterprise Replication collects out-of-row data for transmission after the user transaction has committed. Due to activity on the replicated row, the data might not exist at the time Enterprise Replication collects it for replication. In such cases, Enterprise Replication normally applies a NULL on the target system, unless the data is a smart large object. Therefore, you should avoid placing a NOT NULL constraint on any replicated column that includes out-of-row data.

If a column has smart large objects and the smart large object data does not exist when Enterprise Replication collects it for replication, then Enterprise Replication creates smart large objects with no data and zero size.

Shadow columns

Shadow columns are hidden columns on replicated tables that contain values that are supplied by the database server. The database server uses shadow columns to perform internal operations.

You can add shadow columns to your replicated tables with the CREATE TABLE or ALTER TABLE statement. To view the contents of shadow columns, you must explicitly specify the columns in the projection list of a SELECT statement; shadow columns are not included in the results of SELECT * statements.

The CRCOLS shadow columns, **cdrserver** and **cdrtime**, support conflict resolution. These two columns are hidden shadow columns because they cannot be indexed and cannot be viewed in the system catalog tables. In an update-anywhere replication environment, you must provide for conflict resolution using a conflict resolution rule. When you create a table that uses the time stamp, time stamp plus SPL, or delete wins conflict resolution rule, you must define the shadow columns, **cdrserver** and **cdrtime** on both the source and target replication servers. If you plan to use only the ignore or always-apply conflict resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns for conflict resolution.

The REPLCHECK shadow column, **ifx_replcheck**, supports faster consistency checking. This column is a visible shadow column because it can be indexed and can be viewed in the system catalog table. If you want to improve the performance of the cdr check replicate or cdr check replicateset commands, you can add the **ifx_replcheck** shadow column to the replicate table, and then create an index that includes the **ifx_replcheck** shadow column and your replication key columns.

The ERKEY shadow columns, **ifx_erkey1**, **ifx_erkey2**, and **ifx_erkey3**, are used as the replication key on replicated tables. If you create replicated tables through a grid, these ERKEY columns are automatically added.

Related information

Conflict Resolution on page 40

Preparing Tables for Conflict Resolution on page 77

Shadow column disk space on page 65

Preparing Tables for a Consistency Check Index on page 78

Load and unload data on page 82

Unique key for replication

All tables that are replicated must have a replication key that is composed of one or more columns that uniquely identifies each row. The replication key must be the same on all servers that participate in the replicate. Typically, the replication key is a primary key constraint.

Replicated tables must use a primary key constraint, a unique index or constraint, or the ERKEY shadow columns as the replication key. If your table does not have a primary key or you want to change primary key values while replication is active, you can specify a different key as the replication key. Specify an existing unique index or constraint, or the ERKEY shadow columns as the replication key when you create a replicate. A unique index and a unique constraint are equivalent as replication keys.

If you specify ERKEY columns as the replication key, Enterprise Replication creates a unique index and a unique constraint on the ERKEY columns. The ERKEY columns require storage space.

Important: Because primary key updates are sent as DELETE and INSERT statement pairs, avoid changing the primary key and updating data in the same transaction.

Related information

Preparing tables without primary keys on page 79

Changing the replication key of a replicate on page 168

Cascading Deletes

If a table includes a cascading delete, when a parent row is deleted, the children are also deleted. If both the parent and child tables participate in replication, the deletes for both the parent and child are replicated to the target servers.

If the same table definition exists on the target database, Enterprise Replication attempts to delete the child rows twice. Enterprise Replication usually processes deletions on the parent tables first and then the children tables. When Enterprise Replication processes deletions on the children, an error might result, because the rows were already deleted when the parent was deleted. The table in Table 5: Resolving Cascade Deletes on page 26 indicates how HCL Informix® Enterprise Replication resolves cascading deletes with conflict resolution scopes and rules.

For more information on cascading deletes, see the ON DELETE CASCADE section in the *HCL® Informix® Guide to SQL: Syntax*.

Table 5. Resolving Cascade Deletes

Conflict-Resolution Rule	Conflict-Resolution Scope	Actions on Delete Errors
Time stamp	Row-by-row or transaction	Continue processing rest of the transaction
Delete wins	Row-by-row or transaction	Continue processing rest of the transaction
Ignore	Transaction	Abort entire transaction
Ignore	Row-by-row	Continue processing rest of the transaction

Triggers

A *trigger* is a database object that automatically sets off a specified set of SQL statements when a specified event occurs.

If the **--firetrigger** option is enabled on a replicate, any triggers defined on a table that participates in replication are invoked when transactions are processed on the target server. However, because Enterprise Replication only replicates the final result of a transaction, triggers execute only once on the target regardless of how many triggers execute on the source. In cases where the final evaluation of the transaction results in no replication (for example, an INSERT where the final row image is a DELETE, as shown in Table 2: Insert Followed by a Delete Evaluation Logic on page 15), no triggers execute on the target database.

If the same triggers are defined on both the source and target tables, any insert, update, or delete operation that the triggers generate are also sent to the target database server. For example, the target table might receive replicate data caused by a trigger that also executes locally. Depending on the conflict-resolution rule and scope, these operations can result in errors. To avoid this problem, define the replicate to not fire triggers on the target table.

You might want to design your triggers to take different actions depending on whether a transaction is being performed as part of Enterprise Replication. Use the 'cdrsession' option of the DBINFO() function to determine if the transaction is a replicated transaction. The DBINFO('cdrsession') function returns `1` if the thread performing the database operation is an Enterprise Replication apply or sync thread; otherwise, the function returns `0`.

For more information on triggers, see Enabling Triggers on page 107 and the CREATE TRIGGER section in *HCL® Informix® Guide to SQL: Syntax*.

Related information

DBINFO Function on page

Constraint and replication

When you use constraints, ensure that the constraints you add at the target server are not more restrictive than the constraints at the source server. Discrepancies between constraints at the source and target servers can cause some rows to fail to be replicated.

If your replicated tables that have referential integrity constraints between them, synchronization the data through the replicate set. For replicate sets, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

When you synchronize data, rows that fail to be repaired due to discrepancies between constraints are recorded in the ATS and RIS files.

Sequence Objects

In bi-directional Enterprise Replication, if you replicate tables using sequence objects for update, insert, or delete operations, the same sequence values might be generated on different servers at the same time, leading to conflicts.

To avoid this problem, define sequence objects on each server so that the ranges of generated sequence values are distinct. For more information about the CREATE SEQUENCE and ALTER SEQUENCE statements of SQL, see the *HCL® Informix® Guide to SQL: Syntax*.

The NLSCASE database property

Enterprise Replication supports both case-sensitive databases and NLSCASE INSENSITIVE databases. (Databases created with the NLSCASE INSENSITIVE option ignore letter case in operations on NCHAR and NVARCHAR strings, and on strings of other character data types that are cast explicitly or implicitly to NCHAR or NVARCHAR data types.)

The database server does not prevent a case-sensitive database from being replicated by a database that has the NLSCASE INSENSITIVE property, nor the replication of an NLSCASE INSENSITIVE database by a case-sensitive database. No warning or exception is issued by the database server in either of these cases when you define replication participants.

These two types of database behave differently, however, in operations that classify NCHAR and NVARCHAR strings as duplicates or as distinct values, if the character strings that are being compared differ only in letter case. It is the user's responsibility to make sure that replication participants with different NLSCASE attributes will not cause exceptions or unexpected behavior when replicating the results of operations like the following on NCHAR or NVARCHAR data:

- sorting and collation
- foreign key and primary key dependencies
- enforcing unique constraints
- clustered indexes
- access-method optimizer directives
- queries with WHERE predicates
- queries with UNIQUE or DISTINCT specifications in the projection clause
- queries with ORDER BY clauses
- queries with GROUP BY clauses
- cascading DELETE operations
- table or index storage fragmentation BY EXPRESSION
- table or index storage fragmentation BY LIST
- data distributions from UPDATE STATISTICS operations

To avoid the risk of consistency problems that can result from differences in case-sensitivity, the following policy might be useful when you define replication pairs:

- Replicate case-sensitive databases only with case-sensitive databases.
- Replicate NLSCASE INSENSITIVE databases only with NLSCASE INSENSITIVE databases.

Related information

Duplicate rows in NLSCASE INSENSITIVE databases on page

Replicating Table Hierarchies

To replicate tables that form a hierarchy, you must define a separate replicate for each table.

If you define a replicate on a super table, Enterprise Replication does not automatically create implicit replicate definitions on the subordinate tables.

Tip: Enterprise Replication does not require that the table hierarchies be identical on the source and target servers.

You must use conflict resolution uniformly for all tables in the hierarchy. In other words, either no conflict resolution for all tables or conflict resolution for all tables.

Replication and data types

Enterprise Replication supports built-in data types and user-defined data types, including row types and collection types.

Enterprise Replication does not support replication of simple large objects stored on optical devices.

If you use SERIAL, SERIAL8, or BIGSERIAL data types, you must be careful when defining serial columns.

For non-master replicates, Enterprise Replication does not verify the data types of columns in tables that participate in replication. The replicated column in a table on the source database server must have the same data type as the corresponding column on the target server. The exception to this rule is cross-replication between simple large objects and smart large objects. By using master replicates, you can verify that all participants in a replicate have columns with matching data types. Master replicates also allow verification that each participant contains all replicated columns, and optionally that column names are the same on each participant.

Related information

Flexible Architecture on page 8

Serial data types and replication keys on page 29

Replicating on Heterogeneous Hardware

Enterprise Replication supports all primitive data types across heterogeneous hardware. If you define a replicate that includes non-primitive data types (for example, BYTE and TEXT data), the application must resolve data-representation issues that are architecture dependent.

If you use floating-point data types with heterogeneous hardware, you might need to use IEEE floating point or canonical format for the data transfers. For more information, see Using the IEEE Floating Point or Canonical Format on page 107.

Serial data types and replication keys

You can use a serial data type as a replication key, but you must ensure that the values are unique across all replication servers.

If you plan to use serial data types (SERIAL, SERIAL8, or BIGSERIAL) as the replication key for a table, the same serial value might be generated on two servers at the same time. Use the CDR_SERIAL configuration parameter to generate non-overlapping values for serial columns across all database servers in your replication environment. Set CDR_SERIAL in the `onconfig` file for each primary source database server in the replication system.

You do not need to set the CDR_SERIAL configuration parameter if your replication key has multiple columns and the other columns identify the server on which each row is created.

Related reference

Set configuration parameters for replication on page 72

CDR_SERIAL Configuration Parameter on page 514

Related information

Replication and data types on page 29

Replication of TimeSeries data types

You can replicate tables that have columns with **TimeSeries** data types. You must prepare all replication servers and create time series instances before you create replicates that include **TimeSeries** columns.

Server preparation

All database servers must run Informix® version 12.10 or later.

Before you create a replicate, do the following tasks on all replication servers that will participate in replicating time series data:

1. Set the CDR_TSINSTANCEID configuration parameter to a different value on every replication server to ensure that time series instance IDs do not overlap. You cannot replicate time series instances that were created before you set the CDR_TSINSTANCEID configuration parameter.
2. Create containers that have the same names on all replication servers. You cannot use containers that are created automatically or rolling window containers. If you add containers after replication is set up, add containers with the same names on all replication servers at the same time. The containers can be in different locations on each server.
3. Create the same time series calendars that have the same names on all replication servers.
4. Create time series tables on all replication servers. You cannot use the option to automatically create replicated tables when you define a replicate or template. You cannot nest a **TimeSeries** data type within a **TimeSeries** data type.
5. Create time series instances. You must specify the container name.

Tip: You can quickly set up your replication servers by doing all but the first of these steps through a grid, however, all grid servers must be running Informix® version 12.10 or later.

Rules for defining a replicate

You must follow these rules when you define a replicate for a table that contains a **TimeSeries** column:

- The replicate must be a mastered replicate.
- The Projection list in the participant definition must include all columns in the table.
- The WHERE clause in the participant definition cannot include a **TimeSeries** column.

- You cannot define a participant as send-only.
- The conflict resolution rule must be always-apply.
- The replication key cannot include an opaque data type.
- You cannot enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets.
- You cannot use the --autocreate option to create tables that have **TimeSeries** columns. You must create time series tables on all servers before you define replicates.
- You cannot generate ATS or RIS files in XML format. ATS and RIS files must be in text format.

Restrictions

You cannot run a shared query on a table that includes a **TimeSeries** column. You can, however, run grid queries on a virtual table that is based on a table that has a **TimeSeries** column.

You cannot use the following commands on replicates that include **TimeSeries** columns:

- cdr alter
- cdr remaster
- cdr start sec2er
- cdr swap shadow

You cannot use the following options when you check or repair inconsistencies on a replicate that includes a **TimeSeries** column:

- The --deletewins option in the cdr check replicate or cdr check replicateset command
- The --extratargetrows=merge option in the cdr sync replicate, cdr sync replicateset, cdr check replicate, or cdr check replicateset command
- The --since option in the cdr check replicate or cdr check replicateset command
- The --timestamp option in the cdr check replicate or cdr check replicateset command
- The --where option with a **TimeSeries** column in the WHERE clause in the cdr check replicate command

Although you can add and index an **ifx_replcheck** column on a replicated table that includes a **TimeSeries** column, the speed of consistency checking is not affected.

Related reference

cdr define replicate on page 342

Replication of large objects

How Enterprise Replication handles simple and smart large objects depends on how the objects are stored.

Enterprise Replication replicates the following types of large objects:

- Simple large object data types (TEXT and BYTE)

  You can store simple large objects either in the tblspace with the rest of the table columns (in a dbspace) or in a blobspace. Simple large objects in tblspaces are logged in the logical log and therefore, Enterprise Replication can evaluate the data for replication directly.
- Smart large object data types (BLOB and CLOB)

  You must store smart large objects in sbspaces. Enterprise Replication cannot evaluate large object data that is stored in a blobspace or sbspace; instead, Enterprise Replication uses information about the large object that is stored in the row to evaluate whether the objects need to be replicated.

By default, Enterprise Replication does not include columns that contain unchanged large objects in replicated rows.

Enterprise Replication does not include columns that contain unchanged large objects in replicated rows.

Enterprise Replication allows cross-replication between simple large objects and smart large objects. For example, you can replicate a simple large object on the source database server to a smart large object on the target server or vice versa.

If Enterprise Replication processes a row and discovers undeliverable large object columns, the following actions can occur:

- Any undeliverable columns are set to NULL if the replication operation is an INSERT and the row does not already exist at the target.
- The old value of the local row is retained if the replication operation is an UPDATE or if the row already exists on the target.

Replicating Simple Large Objects from Tblspaces

Enterprise Replication evaluates simple large object data that is stored in a tblspace independently from the rest of its row.

Simple large object data that is stored in tblspaces (rather than in blobspaces) is placed in the logical log. Enterprise Replication reads the logical log to capture and evaluate the data for potential replication.

By default, Enterprise Replication performs time stamp and delete wins conflict detection and resolution at the row level. However, in some cases, simple large object data that is stored in a tblspace (rather than in a blobspace) is accepted by the target server even if the row is rejected.

For simple large objects, if the column on the target database server is also stored in a tblspace, Enterprise Replication evaluates the values of the shadow columns, **cdrserver** and **cdrtime**, in the source and target columns and uses the following logic to determine if the data is to be applied:

- If the column of the replicated data has a time stamp that is greater than the time stamp of the column on the local row, the data for the column is accepted for replication.
- If the server ID and time stamp of the replicated column are equal to the server ID and time stamp on the column on the local row, the data for the column is accepted for replication.
- If there is no SPL conflict-resolution rule and the time stamps are equal, then Enterprise Replication applies the data to the row with the lowest CDR server ID.

If you use the SPL conflict resolution, simple large objects that are stored in tblspaces are handled differently than large objects in blobspaces.

Related information

Delete wins conflict resolution rule on page 48

Time stamp conflict resolution rule on page 42

SPL Conflict Resolution for Large Objects on page 46

Replication of large objects from blobspaces or sbspaces

Enterprise Replication retrieves the large object data directly from the blobspace or sbspace and then sends the data to the target database server.

It is possible that a transaction subsequent to the transaction that is being replicated can modify or delete a simple or smart large object that Enterprise Replication is trying to retrieve. If Enterprise Replication encounters a row whose large object (simple or smart) was modified or deleted by a subsequent transaction, Enterprise Replication does not send the data in the large object. In most cases, the subsequent transaction that modified or deleted the large object is also replicated, so the data again becomes consistent when that transaction is replicated. The data in the large object is inconsistent for only a short time.

The following conditions apply to replicating large objects that are stored in blobspaces or sbspaces:

- Enterprise Replication does not support replication of large object updates performed outside of a row update.
- After you update a large object that is referenced explicitly in the table schema, you must update the referencing row before Enterprise Replication can replicate the updated smart large object. For example:

  ```
  UPDATE table_name SET large_object_column = x
  ```
- Enterprise Replication replicates updates to in-place smart large objects by sending a new copy of the entire smart large object. Enterprise Replication does not send only the logged changes to update smart large objects.
- Enterprise Replication does not support sharing out-of-row data (multiple references to a large object) during replication. If you try to replicate multiple references to the same large object on the source database server, Enterprise Replication does not re-create those references on the target database server. Instead, Enterprise Replication creates multiple large objects on the target database server.

Related information

SPL Conflict Resolution for Large Objects on page 46

Replication of opaque user-defined data types

Opaque data types can be replicated, but have certain restrictions.

You must install and register UDTs and their associated support routines on all database servers that participate in Enterprise Replication before starting replication. If you combine Enterprise Replication with high-availability clusters, you must install UDTs on both the primary and secondary database servers, but only register them on the primary database server.

UDT support functions

If you plan to replicate opaque UDTs, the UDT designer must provide the following types of support functions:

- The streamwrite() and streamread() functions

  The purpose of these functions is similar to the existing send() and receive() functions provided for client/server transmissions. For information about writing these support functions, see the section on Enterprise Replication stream support functions in the *HCL® Informix® DataBlade® API Programmer's Guide*.

  When a row that includes any UDT columns to queue to the target system is prepared for replication, Enterprise Replication calls the streamwrite() function on each UDT column. The function converts the UDT column data from the in-server representation to a representation that can be sent over the network. Enterprise Replication replicates the column without understanding the internal representation of the UDT.

  On the target server, Enterprise Replication calls the streamread() function for each UDT column that it transmitted by the streamwrite() function.
- The compare() function and its supporting greaterthan(), lessthan(), and equal() functions

  Enterprise Replication uses comparison functions to determine whether a replicated column is altered. For example, the comparison functions are used when the replicate definition specifies to replicate only changed columns instead of full rows.

  When you define a compare() function, you must also define the greaterthan(), lessthan(), equal(), or other functions that use the compare() function.

  For more information about writing these support functions, see the *HCL® Informix® User-Defined Routines and Data Types Developer's Guide*.

Requirements during replication

The following requirements apply to replicating opaque data types:

- The WHERE clause of the SELECT statement of the participant modifier can reference an opaque UDT if the UDT is always stored in row.
- Any UDRs in a WHERE clause can use only parameters whose values can be extracted fully from the logged row images, plus any optional constants.
- All of the columns in the SELECT statement of each participant definition must be actual columns in that table. Enterprise Replication does not support virtual columns (results of UDRs on table columns).

- You cannot use SPL routines for conflict resolution if the replicate includes any UDTs in the SELECT statement or if the replicate is defined to replicate only changed columns.
- You can define replicates on tables that contain one or more UDT columns as the replication key.

Replication system design

When you design a replication system, you make three main decisions: how the information flows between servers, how to resolve conflicts between replicated data, and the topology of the network of servers.

Primary-Target Replication System

In the primary-target replication system, the flow of information is in one direction.

In primary-target replication, all database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.

A primary-target replication system can provide one-to-many or many-to-one replication:

- One-to-many replication

  In one-to-many (*distribution*) replication, all changes to a primary database server are replicated to many target database servers. Use this replication model when information gathered at a central site must be disseminated to many scattered sites.

- Many-to-one replication

  In many-to-one (*consolidation*) replication, many primary servers send information to a single target server. Use this replication model when many sites are gathering information (for example, local field studies for an environmental study) that needs to be centralized for final processing.

Related reference

Participant and participant modifier on page 257

Related information

Flexible Architecture on page 8

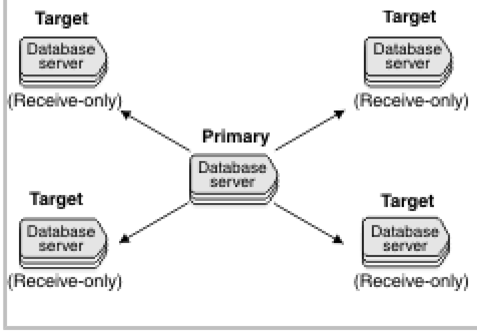
Participant definitions on page 100

Primary-Target Data Dissemination

Data dissemination supports business needs where data is updated in a central location and then replicated to servers that only receive data and do not update data.

This method of distribution can be useful for online transaction processing (OLTP) systems where data is required at several sites, but because of the large amounts of data, read/write capabilities at all sites would slow the performance of the application. The following figure illustrates data dissemination.

Figure 6. Data Dissemination in a Primary-Target Replication System



You specify that a server only receives information when you define the participant for the server as part of the replicate definition. You can specify that all replicates on a server only receive information when you modify the server definition.

Related reference

cdr modify server on page 421

Participant and participant modifier on page 257

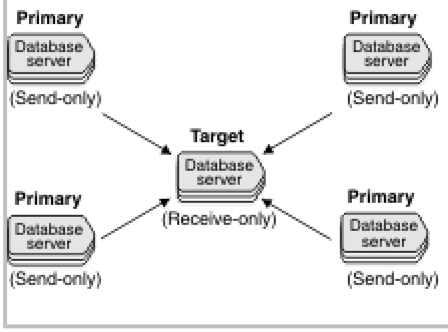
Data consolidation

Businesses can choose to consolidate data into one or more central database servers.

Data consolidation allows the migration of data from several database servers to a central database server. For example, several retail stores can replicate inventory and sales information to headquarters. The retail stores do not need information from other stores but headquarters needs the total inventory and sales of all stores.

In the following figure, the remote locations only send data while a single central database server only receives data.

Figure 7. Data Consolidation in a Primary-Target Replication System



You can also use data consolidation to replicate data from many database servers to more than one central database server. For example, a retail chain has two central database servers, one for the eastern half of the United States, and one for the western half of the United States. The retail stores replicate data to their designated central server and the two central servers replicate data to each other. In this configuration, the replication servers in the retail stores only send data to the central servers, but the central servers both send and receive data.

Businesses can use data consolidation to replicate OLTP data to a dedicated computer for decision support (DSS) analysis. For example, data from several OLTP systems can be replicated to a DSS system for read-only analysis.

The replication key for every replicated row must be unique among the multiple primary database servers.

You specify that a server only sends information when you define the participant for the server as part of the replicate definition. You can specify that all replicates on a server only send information when you modify the server definition.

Related reference

cdr modify server on page 421

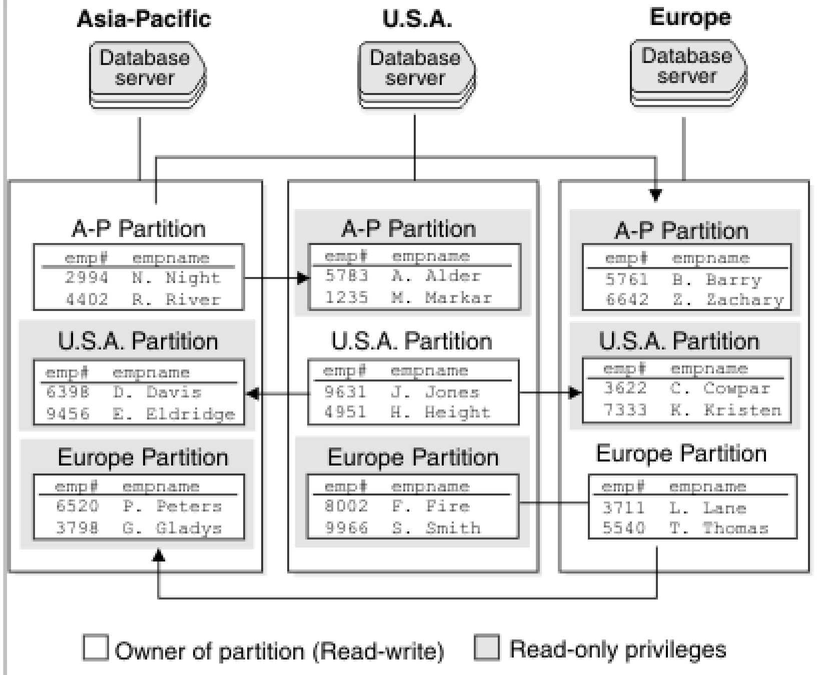
Participant and participant modifier on page 257

Workload Partitioning

Workload partitioning gives businesses the flexibility of assigning data ownership at the table-partition level, rather than within an application.

The following figure illustrates workload partitioning.

Figure 8. Workload Partitioning in a Primary-Target Replication System



emp#	empname
2994	N. Night
4402	R. River

emp#	empname
6398	D. Davis
9456	E. Eldridge

emp#	empname
6520	P. Peters
3798	G. Gladys

emp#	empname
5783	A. Alder
1235	M. Markar

emp#	empname
9631	J. Jones
4951	H. Height

emp#	empname
8002	F. Fire
9966	S. Smith

emp#	empname
5761	B. Barry
6642	Z. Zachary

emp#	empname
3622	C. Cowpar
7333	K. Kristen

emp#	empname
3711	L. Lane
5540	T. Thomas

The replication model matches the partition model for the **employee** tables. The Asia-Pacific database server owns the partition and can therefore update, insert, and delete employee records for personnel in its region. The changes are then propagated to the US and European regions. The Asia-Pacific database server can query or read the other partitions locally, but cannot update those partitions locally. This strategy applies to other regions as well.

Workflow Replication

Unlike the data dissemination model, in a workflow-replication system, the data moves from site to site. Each site processes or approves the data before sending it on to the next site.

Figure 9. A Workflow-Replication System Where Update Authority Moves From Site to Site

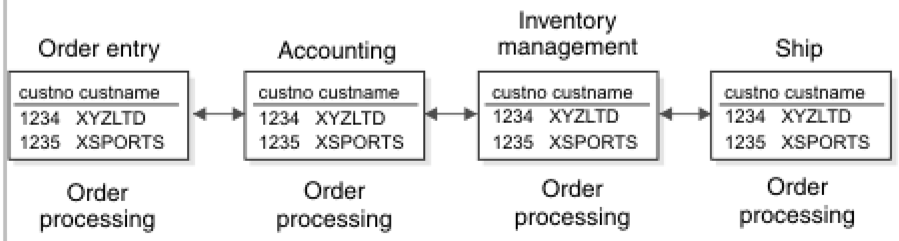


Figure 9: A Workflow-Replication System Where Update Authority Moves From Site to Site on page 38 illustrates an order-processing system. Order processing typically follows a well-ordered series of steps: orders are entered, approved by accounting, inventory is reconciled, and the order is finally shipped.

In a workflow-replication system, application modules can be distributed across multiple sites and databases. Data can also be replicated to sites that need read-only access to the data (for example, if order-entry sites want to monitor the progress of an order).

A workflow-replication system, like the primary-target replication system, allows only unidirectional updates. Many facts that you need to consider for a primary-target replication system should also be considered for the workflow-replication system.

However, unlike the primary-target replication system, availability can become an issue if a database server goes down. The database servers in the workflow-replication system rely on the data updated at a previous site. Consider this fact when you select a workflow-replication system.

Related information

Controlling the replication of large objects on page 112

Primary-Target Considerations

Consider the following factors when you select a primary-target replication system:

- Administration

  Primary-target replication systems are the easiest to administer because all updates are unidirectional and therefore, no data update conflicts occur. Primary-target replication systems use the *ignore* conflict-resolution rule. See Conflict resolution rule on page 40.

- Capacity planning

  All replication systems require you to plan for capacity changes. For more information, see Preparing Data for Replication on page 75.

- High-availability planning

  In the primary-target replication system, if a target database server or network connection goes down, Enterprise Replication continues to log information for the database server until it becomes available again. If a database server is unavailable for some time, you might want to remove the database server from the replication system. If the unavailable database server is the read-write database server, you must plan a course of action to change read-write capabilities on another database server.

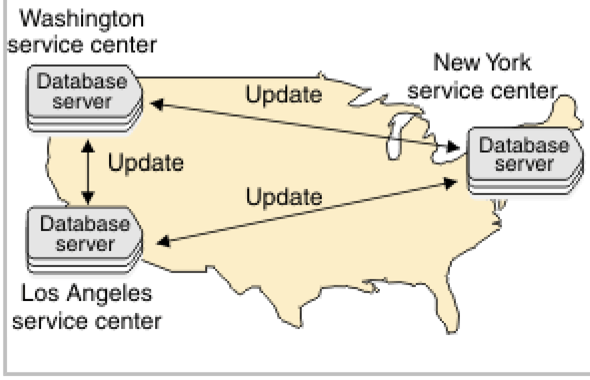
  If you require a fail-safe replication system, you should select a high-availability replication system. For more information, see High-availability replication systems on page 87.

Update-Anywhere Replication System

In update-anywhere replication, changes made on any participating database server are replicated to all other participating database servers. This capability allows users to function autonomously even when other systems or networks in the replication system are not available.

The following figure illustrates an update-anywhere replication system where the service centers in Washington, New York, and Los Angeles each replicate changes to the other two servers.

Figure 10. Update-Anywhere Replication System



Because each service center can update a copy of the data, conflicts can occur when the data is replicated to the other sites. To resolve update conflicts, Enterprise Replication uses conflict resolution.

Review the following information before you select your update-anywhere replication system:

- Administration

  Update-anywhere replication systems allow peer-to-peer updates, and therefore require conflict-resolution. Update-anywhere replication systems require more administration than primary-target replication systems.

- Information consistency

  Some risk is associated with delivering consistent information in an update-anywhere replication system. You determine the amount of risk based on the type of conflict-resolution rules and routines you choose for resolving conflicts. You can configure an update-anywhere replication system where no data is ever lost; however, you might find that other factors (for example, performance) outweigh your need for a fail-safe mechanism to deliver consistent information.

- Capacity Planning

  All replication systems require you to plan for capacity changes and prepare the data for replication. If you choose a time-based conflict resolution rule, you need to provide space for delete tables and add shadow columns to replicated tables.

- High Availability

  If any of your database servers are critical, consider using high-availability clusters to provide backup servers.

Related information

Disk Space for Delete Tables on page 64

Shadow column disk space on page 65

Preparing Data for Replication on page 75

High-availability replication systems on page 87

Specifying Conflict Resolution Rules and Scope on page 104

Flexible Architecture on page 8

Conflict Resolution

When multiple database servers try to update the same row simultaneously (the time stamp for both updates is the same GMT time), a collision occurs. For more information, see Applying replicated data on page 17. Enterprise Replication must determine which new data to replicate. To solve conflict resolution, you must specify the following for each replicate:

- A conflict-resolution rule
- The scope of the rule

Related reference

Replicate only changed columns on page 106

cdr define replicate on page 342

Related information

Shadow columns on page 25

Time synchronization on page 74

Conflict resolution rule

The conflict resolution rule determines how conflicts between replicated transactions are resolved.

Enterprise Replication supports the following conflict resolution rules.

Conflict Resolution Rule	Effect
Ignore	Enterprise Replication does not attempt to resolve conflicts.
Time stamp	The row or transaction with the most recent time stamp is applied.
SPL routine	Enterprise Replication uses a routine written in SPL (Stored Procedure Language) that you provide to determine which data is applied.
Time stamp with SPL routine	If the time stamps are identical, Enterprise Replication uses an SPL routine that you provide to resolve the conflict.
Delete wins	DELETE and INSERT operations win over UPDATE operations; otherwise the row or transaction with the most recent time stamp is applied.
Always-apply	Enterprise Replication does not attempt to resolve conflicts. You must use the always-apply rule when you replicate **TimeSeries** data types.

Related information

Specifying Conflict Resolution Rules and Scope on page 104

Creating replicated tables through a grid on page 136

Ignore Conflict-Resolution Rule

The ignore conflict-resolution rule does not attempt to detect or resolve conflicts.

A row or transaction either applies successfully or it fails. A row might fail to replicate because of standard database reasons, such as a deadlock situation, when an application locks rows. Use the ignore conflict-resolution rule only with a primary-target replication system. If you use ignore with an update-anywhere replication system, your data might become inconsistent.

The ignore conflict-resolution rule can be used only as a primary conflict- resolution rule and can have either a transaction or a row scope (as described in Conflict Resolution Scope on page 50).

The following table describes how the ignore conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 6. Ignore Conflict-Resolution Rule

Row Exists in Target?	INSERT	UPDATE	DELETE
No	Apply row	Discard row	Discard row
Yes	Discard row	Apply row	Apply row

When a replication message fails to apply to a target, you can spool the information to one or both of the following directories:

- Aborted-transaction spooling (ATS)

  If selected, all buffers in a failed replication message that compose a transaction are written to this directory.
- Row-information spooling (RIS)

  If selected, the replication message for a row that cannot be applied to a target is written to this directory.

For more information, see Failed Transaction (ATS and RIS) Files on page 199.

Time stamp conflict resolution rule

The time stamp rule evaluates the latest time stamp of the replication against the target and determines how to resolve any conflict.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The time stamp conflict resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

The time stamp resolution rule behaves differently depending on which scope is in effect:

- Row scope

  Enterprise Replication evaluates one row at a time. The row with the most recent time stamp wins the conflict and is applied to the target database servers. If an SPL routine is defined as a secondary conflict-resolution rule, the routine resolves the conflict when the row times are equal.
- Transaction scope

  Enterprise Replication evaluates the most recent row-update time among all the rows in the replicated transaction. This time is compared to the time stamp of the appropriate target row. If the time stamp of the replicated row is more recent than the target, the entire replicated transaction is applied. If a routine is defined as a secondary conflict resolution rule, it is used to resolve the conflict when the time stamps are equal.

A secondary routine is run only if Enterprise Replication evaluates rows and discovers equal time stamps.

If no secondary conflict-resolution rule is defined and the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The following table shows how a conflict is resolved based on the latest time stamp with row scope. The time stamp T_{last_update} (the time of the last update) represents the row on the target database server with the last (most recent) update. The time stamp T_{repl} (the time when replication occurs) represents the time stamp on the incoming row.

Enterprise Replication first checks to see whether a row with the same replication key exists in either the target table or its corresponding delete table.

If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

The following table describes how the time stamp conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 7. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	INSERT	UPDATE	DELETE
No	n/a	Apply row	Apply row (Convert UPDATE to INSERT)	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Discard row
Yes	$T_{last\_update} = T_{repl}$	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.

Important: Do not remove the delete tables that are created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.

To use time stamp conflict resolution for repairing inconsistencies with the cdr check replicate or cdr check replicateset command, include the --timestamp option with the --repair option. If you temporarily stop replication on a server whose replicates use the time stamp conflict resolution rule, disable the replication server with the cdr disable server command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related reference

cdr disable server on page 383

Related information

Conflict Resolution Scope on page 50

Time synchronization on page 74

Delete wins conflict resolution rule on page 48

Replicating Simple Large Objects from Tblspaces on page 32

Repair inconsistencies by time stamp on page 184

SPL Conflict Resolution Rule

You can write an SPL routine as a primary conflict resolution rule or as secondary conflict resolution rule to the time stamp conflict resolution rule.

You have complete flexibility to determine which row prevails in the database when you create an SPL routine for conflict resolution. However, for most users, the time stamp conflict resolution rule provides sufficient conflict resolution. You can also use SPL routine to save information about the transactions that were discarded during conflict resolution.

SPL routines must follow the following guidelines:

- The owner of an SPL routine that is used for conflict resolution must be the same as the owner of the replicated table.
- Routines for conflict resolution must be in SPL. Enterprise Replication does not allow user-defined routines in C or in Java™.
- You cannot use an SPL routine or a time stamp with an SPL routine if the replicate is defined to replicate only changed columns or the replicated table contains any extensible data types. See Replicate only changed columns on page 106.

Enterprise Replication passes the following information to an SPL routine as arguments.

Argument	**Description**
Server name [CHAR(18)]	From the local target row NULL if local target row does not exist
Time stamp (DATETIME YEAR TO SECOND)	From the local target row NULL if local target row does not exist
Local delete-table indicator [CHAR(1)] or Local key delete-row indicator [CHAR(1)]	Y indicates that the origin of the row is the delete table. K indicates that the origin of the row is the replicate-key delete row. If a conflict occurs while a replication key row is being deleted, because the local row with the old key no longer exists, the received key delete row is passed as the local row (using the seventh argument, local row data). The received key insert row is passed to the stored procedure as the replicated row using the eighth argument.
Server name [CHAR(18)]	Of the replicate source
Time stamp (DATETIME YEAR TO SECOND)	From the replicated row
Replicate action type [CHAR(1)]	`I` - insert `D` - delete `U` - update

Argument	Description
Local row data that is returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list
Replicate row data after-image that is returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list

The routine must set the following arguments before the routine can be applied to the replication message.

Argument	Description
An indicator of the database operation to be performed [CHAR(1)]	Same as the replicate action codes with the following additional codes • `A` - Accept the replicated row and apply the column values returned by the SPL routine. For example, if Enterprise Replication receives an insert and the row exists locally, the insert is converted to an update • `S` - Accept the replicated row and apply the column values as received from the other site. For example, if Enterprise Replication receives an insert and the row exists locally, the insert fails at the time Enterprise Replication tries to apply the transaction to the database, and the transaction aborts with an SQL error. • `O` - Discard the replicated row. • `X` - Abort the transaction.
A non-zero integer value to request logging of the conflict resolution and the integer value in the spooling files (INTEGER)	Logging value takes effect only if logging is configured for this replicate.
The columns of the row to be applied to the target table replicate action type in regular SQL format	This list of column values is not parsed if the routine returns one of the following replicate action types: `S`, `O`, or `X`.

You can use the arguments to develop application-specific routines. For example, you can create a routine in which a database server always wins a conflict regardless of the time stamp.

The following list includes some items to consider when you use an SPL routine for conflict resolution:

- Any action that a routine takes as a result of replication does not replicate.
- You cannot use an SPL routine to start another transaction.
- Frequent use of routines might affect performance.

In addition, you must determine when the SPL routine runs:

- An optimized SPL routine is called only when a collision is detected and the row to be replicated fails to meet one of the following two conditions:
    - It is from the same database server that last updated the local row on the target table.
    - It has a time stamp greater than or equal to that of the local row.
- A nonoptimized SPL routine runs every time Enterprise Replication detects a collision. By default, SPL routines are nonoptimized.

For information on specifying that the SPL routine is optimized, see Conflict Options on page 346.

Tip: Do not assign a routine that is not optimized as a primary conflict resolution rule for applications that usually insert rows successfully.

SPL Conflict Resolution for Large Objects

If the replicate is defined with an SPL conflict-resolution rule, the SPL routine must return the desired action for each smart large object (BLOB or CLOB) and simple large object (BYTE or TEXT) column.

When the routine is invoked, information about each large object column is passed to the routine as five separate fields. The following table describes the fields.

Argument	Description
Column size (INTEGER)	The size of the column (if data exists for this column). NULL if the column is NULL.
BLOB flag [CHAR(1)]	For the local row, the field is always NULL. For the replicated row: • D indicates that the large object data is sent from the source database server. • U indicates that the large object data is unchanged on the source database server.
Column type [CHAR(1)]	• P indicates tblspace data. • B indicates blobspace data. • S indicates sbspace data.

Argument	Description
ID of last update server [CHAR(18)]	The ID of the database server that last updated this column for tblspace data. For blobspace data: NULL For sbspace data: NULL
Last update time (DATETIME YEAR TO SECOND)	For tblspace data: The date and time when the data was last updated. For blobspace data: NULL For sbspace data: NULL

If the routine returns an action code of `A`, `D`, `I`, or `U`, the routine parses the return values of the replicated columns. Each large object column can return a two-character field.

The first character defines the desired option for the large object column, as the following table shows.

Value	Function
`C`	Performs a time-stamp check for this column as used by the time-stamp rule.
`N`	Sets the replicate column to NULL.
`R`	Accepts the replicated data as it is received.
`L`	Retains the local data.

The second character defines the desired option for blobspace or sbspace data if the data is found to be undeliverable, as the following table shows.

Value	Function
`N`	Sets the replicated column to NULL.
`L`	Retains the local data (default).
`O`	Aborts the row.
`X`	Aborts the transaction.

Related information

Replicating Simple Large Objects from Tblspaces on page 32

Replication of large objects from blobspaces or sbspaces on page 33

Delete wins conflict resolution rule

The delete wins rule ensures that DELETE and INSERT operations win over UPDATE operations and that all other conflicts are resolved by comparing time stamps.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The delete wins conflict-resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

The delete wins rule is similar to the time stamp rule except that it prevents upsert operations and does not use a secondary conflict resolution rule. The delete wins rule prevents upsert operations that results from an UPDATE operation that is converted to an INSERT operation because the row to update was not found on the target server. An upsert operation can occur if a row is deleted from a target server before an UPDATE operation is processed on that target server or if an UPDATE operation was processed by the target server before the INSERT operation for that row. Depending on your business logic, upsert operations might violate referential integrity.

The delete wins rule prevents upsert operations in the following ways:

- If a row is deleted on a replication server, that row is deleted on all other replication servers, regardless of whether an UPDATE operation to that row occurred after the delete.
- If an UPDATE operation to a row is received before its INSERT operation, the UPDATE operation fails and generates and ATS or RIS file. The INSERT operation succeeds, but results in data inconsistency. To repair the inconsistency, run the cdr check replicate command with the --repair option.

The delete wins rule handles time stamp conflicts differently depending on which scope is in effect:

- Row scope

  Enterprise Replication evaluates one row at a time. The row with the most recent time stamp wins the conflict and is applied to the target database servers.
- Transaction scope

  Enterprise Replication evaluates the most recent row-update time among all the rows in the replicated transaction. This time is compared to the time stamp of the appropriate target row. If the time stamp of the replicated row is more recent than the target, the entire replicated transaction is applied.

If the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The following table shows how a conflict is resolved with the delete wins rule with row scope. The time stamp T_{last_update} (the time of the last update) represents the row on the target database server with the last (most recent) update. The time stamp T_{repl} (the time when replication occurs) represents the time stamp on the incoming row.

Enterprise Replication first checks to see if a row with the same replication key exists in either the target table or its corresponding delete table. If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

The following table describes how the delete wins conflict resolution rule handles INSERT, UPDATE, and DELETE operations that are performed on the source server.

Table 8. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	INSERT	UPDATE	DELETE
No	n/a	Apply row	Discard row and generate and ATS or RIS file	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Apply row
Yes	$T_{last\_update} = T_{repl}$	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.

Important: Do not remove the delete tables that are created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.

To use delete wins conflict resolution for repairing inconsistencies with the cdr check replicate or cdr check replicateset command, include the --timestamp and --deletewins options with the --repair option. Also set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. If you temporarily stop replication on a server whose replicates use the delete wins conflict resolution rule, disable the replication server with the cdr disable server command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related reference

cdr disable server on page 383

CDR_DELAY_PURGE_DTC configuration parameter on page 500

Related information

Replicating Simple Large Objects from Tblspaces on page 32

Time synchronization on page 74

Time stamp conflict resolution rule on page 42

Repair inconsistencies by time stamp on page 184

Always-Apply Conflict-Resolution Rule

The always-apply conflict-resolution rule does not attempt to detect or resolve conflicts.

Unlike with the ignore conflict-resolution rule, replicated changes are applied even if the operations are not the same on the source and target servers. If a conflict occurs, the current row on the target is deleted and replaced with the replicated row from the source. Use the always-apply conflict-resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent.

The following table describes how the always-apply conflict-resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 9. Always-Apply Conflict-Resolution Rule

Row exists in target?	INSERT	UPDATE	DELETE
No	Apply row	Apply row (convert UPDATE to INSERT)	Apply row (no error returned)
Yes	Apply as an UPDATE (overwrite the existing row)	Apply row	Deletes the row

Conflict Resolution Scope

Each conflict-resolution rule behaves differently depending on the scope.

Enterprise Replication uses the following scopes:

- Row scope

  When you choose a row scope, Enterprise Replication evaluates one row at a time. Only replicated rows that win the conflict resolution with the target rows are applied. If an entire replicated transaction receives row-by-row evaluation, some replicated rows are applied while other replicated rows might not be applied. Row scope can result in fewer failures than transaction scope.

- Transaction scope

  When you choose a transaction scope, Enterprise Replication applies the entire transaction if the replicated transaction wins the conflict resolution. If the target wins the conflict (or other database errors are present), the entire replicated transaction is not applied.

  A transaction scope for conflict resolution guarantees transactional integrity.

Enterprise Replication defers some constraint checking on the target tables until the transaction commits. If a unique constraint or foreign-key constraint violation is found on any row of the transaction at commit time, the entire transaction is rejected (regardless of the scope) and, if you have ATS set up, written to the ATS directory.

Transaction and row scopes are only applicable for apply failure related to conflict resolution, such as missing rows or newer local rows. For other errors, such as lock timeouts, constraint problems, lack of disk space, and so on, the whole incoming transaction is aborted, rolled back, and spooled to ATS or RIS files if so configured, regardless of whether row scope is defined.

Related reference

cdr define replicate on page 342

Related information

Failed Transaction (ATS and RIS) Files on page 199

Time stamp conflict resolution rule on page 42

Specifying Conflict Resolution Rules and Scope on page 104

Choosing a Replication Network Topology

Enterprise replication *topology* describes connections that replication servers make to interact with each other. This topology is the route of replication data (message) transfer from server to server over the network. The replication topology is not synonymous with the physical network topology. Replication server definitions create the replication topology, whereas replicate definitions determine data to be replicated and the sources and destinations within the topology.

The topology that you choose influences the types of replication that you can use. These topics describe the topologies that Enterprise Replication supports.

Related information

Defining Replication Servers on page 93

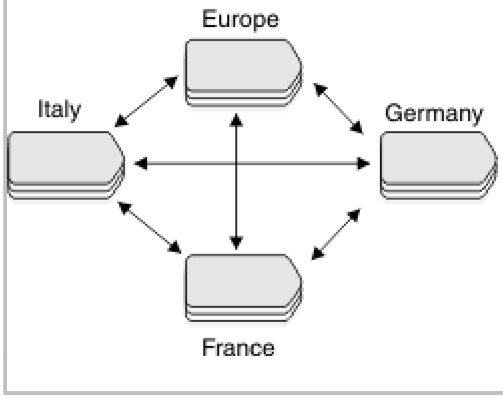
Customizing the Replication Server Definition on page 99

Flexible Architecture on page 8

Fully Connected Topology

Fully connected replication topology indicates that all database servers connect to each other and that Enterprise Replication establishes and manages the connections. Replication messages are sent directly from one database server to another. No additional routing is necessary to deliver replication messages. Figure 11: Fully Connected Topology on page 52 shows a fully connected replication topology. Each database server connects directly to every other database server in the replication environment.

Figure 11. Fully Connected Topology



If necessary, you can also add high-availability clusters and a backup server to any server to provide high availability. For more information, see High-availability replication systems on page 87.

Hierarchical Routing Topology Terminology

Enterprise Replication uses the terms in the Table 10: Replication Topology Terms on page 52 to describe Hierarchical Routing topology.

Table 10. Replication Topology Terms

Term	Definition
Root server	An Enterprise Replication server that is the uppermost level in a hierarchically organized set of information The root is the point from which database servers branch into a logical sequence. All root database servers within Enterprise Replication must be fully interconnected.
Nonroot server	An Enterprise Replication server that is not a root database server but has a complete global catalog and is connected to its parent and to its children
Tree	A data structure that contains database servers that are linked in a hierarchical manner The topmost node is called the root. The root can have zero or more *child* database servers; the root is the *parent* database server to its children.
Parent-child	A relationship between database servers in a tree data structure in which the parent is one step closer to the root than the child.
Leaf server	A database server that has a limited catalog and no children.

A *root* server is fully connected to all other root servers. It has information about all other replication servers in its replication environment. Figure 11: Fully Connected Topology on page 52 shows an environment with four root servers.

A *nonroot server* is similar to a root server except that it forwards all replicated messages for other root servers (and their children) through its parent. All nonroot servers are known to all root and other nonroot servers. A nonroot server might or might not have children. All root and nonroot servers are aware of all other servers in the replication environment.

Important: In *Hierarchical Routing* topologies, Enterprise Replication specifies the synchronization server as the new server's parent in the current topology. For more information, see Customizing the Replication Server Definition on page 99 and cdr define server on page 357.

Related reference

The syscdrs Table on page 590

Related information

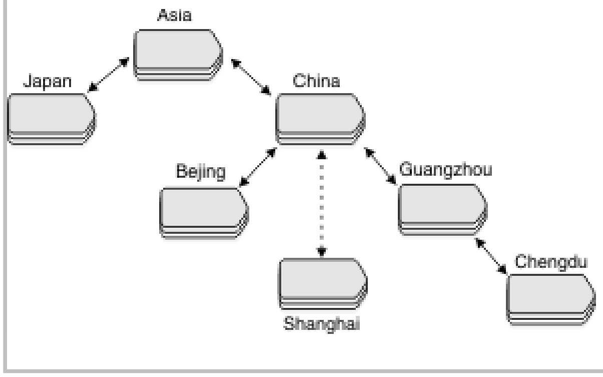
Creating sqlhost group entries for replication servers on page 58

Hierarchical Tree Topology

A *hierarchical tree* consists of a root database server and one or more database servers organized into a tree topology.

The tree contains only one root, which has no parent. Each database server within the tree references its parent. A database server that is not a parent is a leaf. Figure 12: Hierarchical Tree Topology on page 53 illustrates a replication tree.

Figure 12. Hierarchical Tree Topology



In Figure 12: Hierarchical Tree Topology on page 53, the parent-child relationship within the tree is as follows:

- **Asia** is the parent of **China** and **Japan**.
- **China** is the child of **Asia** *and* the parent of **Beijing**, **Shanghai**, and **Guangzhou**.
- **Guangzhou** is the child of **China** *and* the parent of **Chengdu**.

Asia is the root database server. **Japan**, **China**, and **Guangzhou** are nonroot database servers. You can define **Beijing**, **Shanghai**, and **Chengdu** as either nonroot database servers or leaf database servers, depending on how you plan to use them. The dashed connection from **China** to **Shanghai** indicates that **Shanghai** is a leaf server.

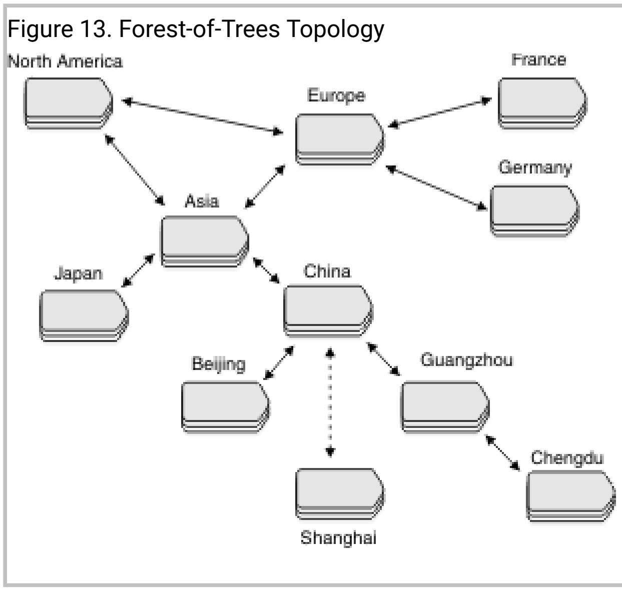
You can define a replicate that replicates data exclusively between **Shanghai** and **Japan**. However, the transaction data would must go through **China** and **Asia**. If either **China** or **Asia** is offline replication is suspended. Similarly, a replicate defined between **Japan** and **China** would require **Asia** to be functioning, even though both **Japan** and **China**, as nonroot servers, have entries in their `sqlhosts` files for each other.

Parent servers are good candidates for using high-availability clusters to provide backup servers.

Forest of trees topology

A *forest of trees* consists of several hierarchical trees whose root database servers are fully connected. Each hierarchical tree starts with a root database server. The root database servers transfer replication messages to the other root servers for delivery to its child database servers. Figure 13: Forest-of-Trees Topology on page 54 shows a forest of trees.

Figure 13. Forest-of-Trees Topology



In Figure 13: Forest-of-Trees Topology on page 54, **North America**, **Asia**, and **Europe** are root database servers. That is, they are fully connected with each other. **France** and **Germany** are in a tree whose root is **Europe**. **Asia** is the root for the six database servers in its tree.

In a forest of trees, all replication messages from one tree to another must pass through their roots. For example, a replication message from **Beijing** to **France** must pass through **China**, **Asia**, and **Europe**.

Organizing the database servers in a hierarchical tree or a forest of trees greatly reduces the number of physical connections that are required to make a replication system. If all the database servers in Figure 13: Forest-of-Trees Topology on page 54 were fully connected, instead of being organized in trees, 55 connections would be required.

To ensure that all servers retain access to the replication system, use high-availability clusters on parent servers. For more information, see Using high-availability clusters in a forest of trees topology on page 88.

Setting up and managing Enterprise Replication

After you design your replication system, you define it and start replication.

To set up replication:

1. Select the Enterprise Replication system and network topology to use for your replication environment.
2. Prepare the replication environment.
3. Define database servers for replication.
4. Define a grid and create replicated tables.

After you define and start your replication system, you can monitor and maintain it.

Instead of creating a grid, you can create a replicate set by defining and realizing a template, or you can define replicates and participants and then create a replicate set and start replication.

Preparing the Replication Environment

The following topics explain the steps that are required for setting up Enterprise Replication.

The cdr autoconfig serve command can auto-configure Enterprise Replication for a database server that has a configured storage pool, and propagate connectivity information between the database servers in an Enterprise Replication domain. Complete the following steps to auto-configure Enterprise Replication:

1. Verify that the CDR_AUTO_DISCOVER configuration parameter is set to `1` on all database servers.
2. Verify that the storage pool is configured on any database server that you are adding to the Enterprise Replication Domain.
3. Choose a database server to be your source server for propagating configuration changes to other servers, and for replicating date to a newly added replication server.
4. On the source server, set trusted-host information for all database servers by running the admin() or task() function with the cdr add trustedhost argument.
5. Verify that all replication servers are active.
6. On the source server, run the cdr autoconfig serve command. Alternatively, you can run the cdr autoconfig serve command on a different database server, but you must specify the source server's information in the command.

Related information

Creating a new domain by cloning a server on page 94

Trusted-host information on page

cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API) on page

Preparing the Network Environment

You must prepare the network environment for each database server in an Enterprise Replication domain.

About this task

The following files are involved in configuring the replication network:

- `sqlhosts`: Specifies replication connectivity, including server groups, connection security, and network security.
- `hosts`: Specifies hosts names if you are not using Domain Name Service (DNS).
- `services`: Specifies the service name that is associated with a port number.
- The trusted-hosts file. You specify this file by setting the REMOTE_SERVER_CFG configuration parameter. This file specifies the host names for trusted replication servers.
- If you use Connection Managers for managing connectivity, you must create a Connection Manager configuration file.

You can manually specify `sqlhost` and trusted-host file information to each database server, or you can run the admin() or task() function with the cdr add trustedhost argument to add entries to the trusted host files, and then run the cdr autoconfig serv command to propagate `sqlhost` and trusted-host file entries to other database servers in an Enterprise Replication domain.

To prepare your network environment, configure the following for each replication server:

1. If you are not using DNS, configure replication-server host information in the `hosts` file.
2. Configure port information in the `services` files and the `sqlhosts` files.
3. Create group entries for each replication servers in the `sqlhosts` file.
4. If necessary, configure secure ports for replication servers in the `sqlhosts` file.
5. If necessary, configure network security for client/server communications in the `sqlhosts` files.

   Note: Support for Communication Support Module (CSM) is removed starting Informix Server 14.10.xC9 . You should use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) instead.

6. Create a trusted-host file and add entries for each trusted hosts.

Related reference

cdr autoconfig serv on page 291

CDR_AUTO_DISCOVER configuration parameter on page 498

Related information

Replication Examples on page 596

The syncsqlhosts utility on page

Client/server communication on page

Trusted-host information on page

cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API) on page

Configuring hosts information for replication servers

If you are not using Domain Name Service (DNS) to identify IP addresses and system names, you do need to configure the `hosts` file on each replication server to add the IP addresses and system names for all other replication servers in the domain.

About this task

The `hosts` file is in the following location.

Operating System	File
UNIX™	`/etc/hosts`
Windows™	`%WINDIR%\system32\drivers\etc\hosts`

Important: Leave a blank line at the end of the `hosts` file on Windows™.

For example, your `hosts` file might look like the following:

```
192.168.0.1 ny.usa.com
192.168.0.2 tokyo.japan.com
192.168.0.3 rome.italy.com
192.168.0.4 perth.australia.com
```

Configuring ports and service names for replication servers

Replication servers must know the port numbers for each of the other replication servers in the domain.

About this task

Configure port numbers for replication servers in one of the following ways:

- Specify the port numbers in the `sqlhosts` file. This method risks conflicting with port numbers being used by other applications.
- Specify the service names in the `sqlhosts` file and specify the port numbers for each service name in the `services` file.

The `services` file is in the following location.

Operating System	File
UNIX™	`/etc/services`
Windows™	`%WINDIR%\system32\drivers\etc\services`

Important: Leave a blank line at the end of the `services` file on Windows™.

For example, your `services` file might look like the following:

```
sydney 5327/tcp
melbourne 5327/tcp
```

If the database servers reside on the same system, you must provide unique port numbers for each.

Related reference

cdr start sec2er on page 450

Related information

Verifying the services Files on All Computers on page 610

Creating sqlhost group entries for replication servers

The `sqlhosts` file on the host of each replication server must specify a group entry for each replication server in an Enterprise Replication domain. You can manually specify `sqlhost` file information, or run the cdr autoconfig serv command to add entries to a database server's `sqlhost` file, and then propagate the entries to other database servers in an Enterprise Replication domain. However, if you are configuring secure ports, you cannot use the cdr autoconfig serv command.

Typically, a server group includes only one database server. However, if the computer has multiple network protocols or network interface cards, the server group includes all aliases for the database server. Enterprise Replication treats the server group as one object, whether it includes one or several database server names.

UNIX: You define a database server group in the `sqlhosts` file.

Windows: Use IBM® Informix® Server Administrator (ISA), rather than regedt32, to set up the SQLHOSTS registry key and database server group registry key on your Windows™ system. In addition, ISA allows you to administer your replication system from a web browser. For information about preparing the SQLHOSTS connectivity information on Windows™ systems, see SQLHOSTS Registry Key (Windows) on page 605.

The following example shows `sqlhosts` file entries for four Enterprise Replication servers:

- **serv1**
- **serv2**
- **serv3**
- **serv4**

```
#dbservername  nettype   hostname             servicename  options
 g_serv1       group     -                    -            i=143
 serv1         ontlitcp  ny.usa.com           1230         g=g_serv1

 g_serv2       group     -                    -            i=144
 serv2         ontlitcp  tokyo.japan.com      1231         g=g_serv2

 g_serv3       group     -                    -            i=145
 serv3         ontlitcp  rome.italy.com       1232         g=g_serv3
```

```
g_serv4       group     -                   -          i=146
serv4         ontlitcp  perth.australia.com  1233       g=g_serv4
```

Each server has two entries of information:

- A group definition, which specifies a group name and unique ID for the replication server
- Connectivity information for the database server

All Enterprise Replication commands and options use the name of the database server group or the more familiar database server name (that is, the name that is specified by the **INFORMIXSERVER** environment variable) for all references to database servers. The exception is the --connect option, which can use either a server name or a group name.

Leaf servers in hierarchical routing topologies do not require connectivity information for all replication servers. A leaf server requires connectivity information for only itself and its parent.

Related reference

cdr autoconfig serv on page 291

Related information

Connect Option on page 256

The sqlhosts information on page

Hierarchical Routing Topology Terminology on page 52

sqlhosts connectivity information on page

The syncsqlhosts utility on page

Setting Up Database Server Groups for High-Availability Cluster Servers on page 89

Setting Up the Database Server Group Registry Key on page 608

Configuring secure ports for connections between replication servers

If database servers in your Enterprise Replication environment are on a network that is not trusted, you can configure secure ports and an encrypted password file to enable secure connections.

About this task

The secure ports that are listed in the `sqlhosts` files can be used only for communication between database servers. You must configure a separate port for local client/server communications.

To configure a secure port for replication:

1. In the `sqlhosts` file on each server, create a group entry with two connections for the local server:
    a. Create one connection entry without the s=6 option to configure local communication with utilities, such as the cdr utility and Connection Managers.
    b. Create one connection entry with the s=6 option to configure communication between servers.

   Example

In the following example, the value of the DBSERVERNAME configuration parameter is **serv1**:

```
#dbservername    nettype       hostname      servicename    options
serv1            ontlitcp      ny.usa.com    ertest1
g_serv1          group         -             -              i=143
serv1_s6         ontlitcp      ny.usa.com    ertest10       g=g_serv1,s=6
```

Note: Do not use the cdr autoconfig serv command if you configure secure ports. `sqlhosts` file entries must be manually added if any entries include the s=6 option.

2. In the `sqlhosts` file on each server's host, add entries for each of the other servers in the domain. Use the server names that are associated with the s=6 options.
3. Create a trusted-host file that includes the host names of the other replication servers in the domain, each on a separate line.

   You can manually create the trusted-host file in `$INFORMIXDIR/etc`, and then set the REMOTE_SERVER_CFG configuration parameter to the name of the trusted-host file. Alternatively, you can run the admin() or task() function with the cdr add trustedhost argument to set a replication server's REMOTE_SERVER_CFG configuration parameter and add entries to the server's trusted-host file. If the replication server is part of a high-availability cluster, running the admin() or task() function with the cdr add trustedhost argument propagates trusted-host entries to other database servers in a high-availability cluster.

   Note: You cannot use the `hosts.equiv` trusted-host file when you configure secure ports.

   Example

   The following example trusted-host file has entries for three hosts, and specifies both host names and domain names:

   ```
   #hostname
   tokyo.japan.com
   tokyo

   rome.italy.com
   rome

   perth.australia.com
   perth
   ```

   A database server on a listed host connects to the local database server instance through the `sqlhosts` file entry with the s=6 option.
4. Set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1` in the `onconfig` file.
5. Using a text editor, create and save a password file. The password file includes the host name, alternative server name, user ID, and password for each server and the server group.

   Example

   For example, if the user ID for server **serv1** is `informix`, the alias for the database server that uses a secure port is **serv1_s6**, and the password was `informix_pw`, use the following password file entries:

   ```
   serv1_s6  serv1  informix  informix_pw
   g_serv1  serv1  informix  informix_pw
   ```

6. Encrypt the password file by running the onpassword utility.

   Example

   For example, if you named the text file in step 5 `$INFORMIXDIR/etc/server_passwords`, and you wanted the file encrypted with a key called access_key, use the following command:

   ```
   onpassword -k access_key -e $INFORMIXDIR/etc/server_passwords
   ```

   The encrypted file is saved as: `$INFORMIXDIR/etc/passwd_file`.

   Important: To prevent unauthorized access to the server passwords, remove the unencrypted password file, `$INFORMIXDIR/etc/server_passwords` after you create the encrypted file.

What to do next

If you do not configure a password file, you must run the cdr utility on the local computer, for example:

```
cdr list server --connect=serv1
```

Because secure ports can be used only for replication communication, you cannot test the connections until you start replication.

Related information

Testing the replication network on page 62

S6_USE_REMOTE_SERVER_CFG configuration parameter on page

The onpassword utility on page

The sqlhosts file and the SQLHOSTS registry key on page

Setting up the Replication privilege for non-root owners

DBSERVERALIASES configuration parameter on page

REMOTE_SERVER_CFG configuration parameter on page

Configuring secure connections for grid queries on page 148

Configuring network encryption for replication servers

You encrypt client/server network communication by specifying the ENCCSM module with the communications support module (CSM) option in the `sqlhosts` file. You encrypt Enterprise Replication communication by setting encryption configuration parameter ENCRYPT_SMX or by configuring ER group option for onsocssl port in `sqlhosts` file. Unless onsocssl port is used for communicating with peer ER servers, for communicating with older server versions before 14.10xC6, Enterprise Replication requires configuring ENCRYPT_CDR instead of ENCRYPT_SMX.

You cannot configure an Enterprise Replication connection with a CSM.

To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server. The configuration in the SQLHOSTS file would look like the following example.

```
#dbservername  nettype   hostname    servicename  options
 gserv1        group     -           -            i=143
 serv1         ontlitcp  ny.usa.com  ertest1      g=gserv1
 c_serv1       ontlitcp  ny.usa.com  ertest10     csm=(ENCCSM)
```

In this example, **serv1** and **c_serv1** are two connection ports on the same database server. Encrypted client/server communication uses the **c_serv1** port, while encrypted Enterprise Replication uses the **serv1** port.

For more information on encrypting client/server network communications, see the *HCL® Informix® Administrator's Guide*.

Related reference

Set configuration parameters for replication on page 72

Enterprise Replication configuration parameter and environment variable reference on page 498

Testing the replication network

After you set up the network environment, test the connections between the replication servers. You cannot test a connection that uses the s=6 option in the `sqlhosts` file.

About this task

To test the network environment:

1. Verify the network connection.
   Use the ping command to test the connection between two systems. For example, from **ny.usa.com**, test the connection to **tokyo.japan.com**:

   ```
   ping tokyo.japan.com
   ```

2. Test the trusted environment:
   a. Run **dbaccess** as user **informix** or as the owner if it is a non-root server.
   b. Run **dbaccess**.
   c. Select the **Connection** menu option.
   d. Select the **Connect** menu option.
   e. Connect to the server group name and the server name of the other hosts.

      For example, if you are running **dbaccess** on **ny.usa.com**, and you are testing the connection to a database server on **tokyo.japan.com**, select **serv2** and **g_serv2**.
   f. When prompted for the USER NAME, press `Enter`.

   If you can connect to the host database server, the host server is trusted for user **informix**.

   If you can connect to the host database server, the host server trusts the connection from the remote host as user **informix** or as the owner if the remote host is a non-root server.

   For more information, see the *HCL® Informix® DB-Access User's Guide*.

Related information

Configuring secure ports for connections between replication servers on page 59

Setting up the Replication privilege for non-root owners

Testing the password file

You create and encrypt a password file to allow the CDR utility to access to a secure network environment. Use these steps to test that the encrypted password file is correctly configured.

To test the password file configuration:

Use the cdr view state -c ***remote_server_group_name*** command to verify that the password file supplies the correct password to the CDR command.
For example, if your remote server group was named `g_serv2`, specify the following command:

```
cdr view state -c g_serv2
```

Result
The state of all configured enterprise replication servers is returned. If enterprise replication is not defined, but the password file is set up correctly, the following message is returned:

```
ERROR:ER not defined on g_serv2
```

If the CDR utility is unable to connect to the server or if the following error is returned then verify that `$INFORMIXDIR/etc/passwd_file` is correctly configured.

```
25539: Invalid connection-type
```

The following is an example of command output returned when Enterprise Replication and the password file are correctly configured:

```
$  cdr view state -c g_serv2
STATE
Source   ER               Capture          Network          Apply
         State            State            State            State
---------------------------------------------------------------------------
g_serv2 Active           Running          Running          Running
g_serv1 Active           Running          Running          Running
```

Preparing the Disk

These topics describe how to prepare your disk for Enterprise Replication.

Logical Log Configuration Disk Space

The database server uses the logical log to store a record of changes to the data since the last archive. Enterprise Replication requires the logical log to contain entire row images for updated rows, including deleted rows.

The database server normally logs only columns that have changed. This behavior is called the logical-log record reduction option. Enterprise Replication deactivates this option for tables that participate in replication. (The logical-log record reduction option remains enabled for tables that do *not* participate in Enterprise Replication.) Enterprise Replication logs all columns, not only the columns that have changed, which increases the size of your logical log.

To determine the size of your logical log, examine your data activity for normal operations and for the replication system you defined. Keep in mind that defining replication on a table causes Enterprise Replication to deactivate log reduction for that table, and that your transactions might log more data.

Important: Enterprise Replication performs internal cleanup tasks based on how often the log files switch. If the log files switch too frequently, Enterprise Replication might perform excessive cleanup work.

Logical Log Configuration Guidelines

Logical logs must be configured correctly for Enterprise Replication.

Use the following guidelines when configuring your logical log files:

- Make sure that all logical log files are approximately the same size.
- Make the size of the logical log files large enough so that the database server switches log files no more than once every 15 minutes during normal processing.
- Plan to have sufficient logical-log space to hold at least four times the maximum transaction size.
- Set LTXEHWM (long-transaction, exclusive-access, high-watermark) 30 percent larger than LTXHWM (long-transaction high-watermark).

Important: If you specify that the database server allocate logical log files dynamically (DYNAMIC_LOGS), it is recommended that you set LTXEHWM to no higher than 70 when using Enterprise Replication.

For more information about logical logs and these configuration parameters, see *HCL® Informix® Administrator's Reference* and *HCL® Informix® Administrator's Guide*.

The database server can add logs dynamically when Enterprise Replication approaches a potential log wrap situation if the CDR_MAX_DYNAMIC_LOGS configuration parameter is set to a non-zero integer.

Related information

Handle potential log wrapping on page 214

Preventing Memory Queues from Overflowing on page 212

Disk Space for Delete Tables

If you use the time stamp, time stamp and SPL routine, or delete wins conflict resolution rules, you must provide enough disk space for the *delete tables* that Enterprise Replication creates to keep track of modified rows for conflict resolution.

Delete tables handle conflicts such as when a DELETE or UPDATE operation finds no corresponding row on the target. The DTCleaner thread removes a row from the delete tables after all the servers have progressed beyond that row. Enterprise

Replication does not create delete tables for tables that have replicates defined with a conflict resolution rule of ignore or always-apply.

Delete tables are created on the database server where the data originates and on all the database servers to which data gets replicated. Delete tables are stored in the same dbspaces, using the same fragmentation strategy, as their base tables.

To determine the disk space requirements to accommodate delete tables, estimate how many rows will be deleted or modified. For example, if the base table has 100 megabytes of data, but only half the rows might be deleted or modified, then 50 megabytes is a reasonable estimate for the size of the delete table.

Important: Do not remove the delete tables created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.

Related reference

Replicate only changed columns on page 106

Related information

Update-Anywhere Replication System on page 39

Shadow column disk space

If you plan to use shadow columns, make sure to allow additional disk space for their values.

If you plan to use any conflict-resolution rule except ignore or always-apply, you must allow for an additional 8 bytes for the CRCOLS shadow columns, **cdrserver** and **cdrtime**, which store the server and time stamp information that Enterprise Replication uses for conflict resolution.

If you want to speed consistency checking by indexing the REPLCHECK shadow column, you must allow for an additional 8 bytes for the **ifx_replcheck** shadow column.

If you want to use ERKEY shadow columns as the replication key, or you create your replicated tables through a grid, you must allow of an additional 10 bytes for the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**. When you create replicated tables through a grid, these ERKEY columns are automatically added.ERKEY columns also require disk space for the index that is created on them. In addition to the standard partition and page overhead, for each row in the table the ERKEY index uses 14 bytes for non-fragmented tables and 18 bytes for fragmented tables for each row in the table.

The following table shows the amount of space used by each shadow column.

Table 11. Shadow column size

Shadow column name	Data type	Size
cdrserver	INTEGER	4 bytes

Table 11. Shadow column size (continued)

Shadow column name	Data type	Size
cdrtime	INTEGER	4 bytes
ifx_replcheck	BIGINT	8 bytes
ifx_erkey_1	INTEGER	4 bytes
ifx_erkey_2	INTEGER	4 bytes
ifx_erkey_3	SMALLINT	2 bytes

The shadow columns claim disk space immediately, except when CRCOLS and ERKEY columns are added to an existing table.

Related information

Update-Anywhere Replication System on page 39

Shadow columns on page 25

Preparing Tables for Conflict Resolution on page 77

Preparing Tables for a Consistency Check Index on page 78

Setting Up Send and Receive Queue Spool Areas

The term *data queue* refers to both the *send queue* and the *receive queue*. Enterprise Replication collects information from the logical logs and places the data to be transferred in the send queue. Then Enterprise Replication transfers the contents of the send queue to the receive queue on the target server. Enterprise Replication on the target reads the data from the receive queue and applies the changes to the tables on the target server.

The send and receive queues reside in memory and are managed by the Reliable Queue Manager (RQM). The CDR_QUEUEMEM configuration parameter (CDR_QUEUEMEM Configuration Parameter on page 514) specifies the amount of memory space that is available for the data queues.

When a queue in memory fills (for the receive queue, this only occurs with large transactions), the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of *transaction records* (headers that contain internal information for Enterprise Replication), *replicate information* (summaries of the replication information for each transaction), and *row data* (the actual replicated data). Spooled transaction records and replication records are stored in transaction tables and replication tables in a single dbspace. Spooled row data is stored in one or more sbspaces.

Important: To prevent the send and receive queues from spooling to disk, see Preventing Memory Queues from Overflowing on page 212.

Related information

Send queues and receive queues on page 14

Preventing Memory Queues from Overflowing on page 212

Row Data sbspaces

Replicated data might include UDT and CLOB or BLOB data types. Therefore, the spooled row data is stored as smart large objects in one or more sbspaces.

Important: Before starting Enterprise Replication, you must create at least one sbspace for spooled row data and set the CDR_QDATA_SBSPACE configuration parameter to the name of the sbspace.

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. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_QDATA_SBSPACE configuration parameter when defining a replication server. If the CDR_QDATA_SBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the cdr define command performs the following tasks:

- Creates a new sbspace using one or more new chunks from the storage pool
- Sets the CDR_QDATA_SBSPACE configuration parameter both in memory and in the `onconfig` file to the newly defined sbspace.

For clusters, the cdr define command creates new sbspaces and sets the CDR_QDATA_SBSPACE configuration parameters in all secondary database servers, as well.

Note: A database server's storage pool must have 500 MB of free space for sbspaces, and chunk sizes of 100 MB or greater for the database server to use automatic storage provisioning.

Related reference

Set configuration parameters for replication on page 72

Related information

Defining Replication Servers on page 93

Creating sbspaces for Spooled Row Data

You must create dedicated sbspaces for spooled row data.

About this task

Follow these guidelines when creating sbspaces for spooled row data:

- Create all the sbspaces of same default log mode type with the same size.
- Do not use Enterprise Replication row data sbspaces for non-Enterprise Replication activity.
- Ensure that the sbspaces are sufficiently large.

  To determine the size of your spooled row data sbspaces, determine your log usage and then consider how much data you can collect if your network goes down. For example, assume that you usually log 40 megabytes of data each day, but only 10 percent of that is replicated data. If your network is down for 24 hours and you estimate that 4 MB of replicated data are logged each day, the size of the sbspaces you identify for the spooled row data must be a total of at least 4 MB.

  Windows™ Only

  On Windows™, increase the resulting size of the sbspace by approximately a factor of two. (The default page size, the way that data maps onto a page, and the number of pages written to disk differs on Windows™.)

Important: When the row data sbspaces fill, Enterprise Replication hangs until you either resolve the problem that is causing Enterprise Replication to spool or allocate additional disk space to the sbspaces. For more information, see Preventing Memory Queues from Overflowing on page 212.

To create row data sbspaces, use the **onspaces -c** utility. For example, to create a 4-megabyte sbspace, called **er_sbspace**, using raw disk space on UNIX™ with an offset of 0, enter:

```
onspaces -c -S er_sbspace -p /dev/rdsk/c0t1d0s4 -o 0 -s 4000\
    -m /dev/rdsk2/c0t1d0s4 0 \
    -Df "AVG_LO_SIZE=2,LOGGING=OFF"
```

The path name for an sbspace cannot be longer than 256 bytes.

The **-m** option specifies the location and offset of the sbspace mirror. The **-Df** option specifies default behavior of the smart large objects stored in the sbspace:

- AVG_LO_SIZE (average large object size)

  Set this parameter to the expected average transaction size (in KB). The database server uses this value to calculate the metadata size. The minimum value for AVG_LO_SIZE is 2 KB, which is appropriate for Enterprise Replication in most cases. (The default value of AVG_LO_SIZE is 32 KB.) If you set AVG_LO_SIZE to larger than the expected transaction size, you might run out of metadata space. If you set AVG_LO_SIZE too small, you might waste space on metadata.

- LOGGING

  Set this parameter to OFF to create an sbspace without logging. Set this parameter to ON to create an sbspace with logging. Use a combination of logging and non-logging sbspaces for Enterprise Replication. For more information, see Logging Mode for sbspaces on page 69.

Set the CDR_QDATA_SBSPACE configuration parameter in the ONCONFIG file to the location of the row data sbspace (**er_sbspace**, in this example). For more information, see CDR_QDATA_SBSPACE Configuration Parameter on page 512.

Logging Mode for sbspaces

About this task

Enterprise Replication uses the default log mode that the sbspace was created with for spooling row data.

Create sbspaces with a default logging mode of ON or OFF according to the types of transactions Enterprise Replication replicates:

- **LOGGING=ON**

  Create sbspaces with LOGGING set to ON to support these situations:

  - Replicated systems with high-availability clusters

    Enterprise Replication must use logging sbspaces for transactions involved in high-availability clusters.

  - Small transactions

    Enterprise Replication uses logging sbspaces for transactions that are less than a page size (2K or 4K) of replicated data.

  For logging sbspaces, performance might be enhanced because logging mode enables asynchronous IO. However, a logging sbspace consumes additional logical-log space.

- **LOGGING=OFF**

  Create sbspaces with LOGGING set to OFF to support replication of large transactions (greater than a page size of replicated data).

  It is recommended that you mirror non-logging sbspaces. For more information, see the chapter on managing disk space in the *HCL® Informix® Administrator's Guide* and the *HCL® Informix® Administrator's Reference*.

  For non-logging sbspaces, performance is enhanced on the database server when Enterprise Replication spools to disk because Enterprise Replication writes less data to disk.

Important: Do not change the Enterprise Replication sbspace default log mode while Enterprise Replication is running. To change the default log mode, follow the procedure below.

You can change the default logging mode of the row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To change the default logging mode of a row data sbspace:

1. Shut down the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in an sbspace.
5. Change the default logging mode for the sbspace.
6. Add the sbspace name to the CDR_QDATA_SBSPACE configuration parameter value list.
7. Shut down and restart the database server using the **onmode -ky** and **oninit** commands.

Dropping a Spooled Row Data sbspace

About this task

Important: Do not drop an Enterprise Replication row data sbspace using the **onspaces -d -f** (force) command.

You can drop a row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To drop a row data sbspace

1. Shutdown the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in a sbspace.
5. If the sbspace was added from the storage pool, use the drop sbspace to storagepool argument with the admin() or task() function to return the empty sbspace to the storage pool.

Related information

drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API) on page

Setting Up the Grouper Paging File

Enterprise Replication uses a grouper paging mechanism for evaluating large transactions. A transaction is large if the portion to be replicated meets at least one of the following conditions:

- It has greater than 5,000 log records.
- It exceeds one fifth the size of the value of the CDR_QUEUEMEM ONCONFIG variable.
- It exceeds one tenth the size of the value of the SHMVIRTSIZE configuration variable.

The location of the sbspace used for the paging file is determined by the first of the following ONCONFIG configuration parameters that is set:

- SBSPACETEMP
- SBSPACENAME
- CDR_QDATA_SBSPACE

The best solution is to set up an unlogged sbspace, as specified by the SBSPACETEMP configuration parameter. All updates to the paging files are unlogged.

The size of the paging sbspace should be at least three times the size of the largest transaction to be processed. This sbspace is also used by the database server for other tasks; consider its overall usage when determining size requirements.

❗ **Important:** If the paging sbspace fills, Enterprise Replication hangs until you allocate additional disk space to the sbspace. If additional space is unavailable, use the **cdr stop** command to stop replication.

Creating ATS and RIS directories

You can create directories for Aborted Transactions Spooling (ATS) and Row Information Spooling (RIS) files instead of using the default directories.

About this task

ATS and RIS files contain information about failed transactions and aborted rows. You can repair data after a replicated transaction fails by applying ATS and RIS files. Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied. ATS and RIS files are relevant only if you specify a conflict resolution role other than ignore or always-apply.

The default location for ATS and RIS directories is `/tmp` (UNIX™) or `\tmp` (Windows™).

If you want to use non-default directories, create the ATS or RIS directories before you define the server for replication. The path names for the ATS and RIS directories cannot be longer than 256 characters.

Choose from:

- Create RIS directories on all replication servers in the domain.
- Create ATS directories on all replication servers in the domain, if you are using update-anywhere replication.
- Create the ATS directory on the target system, if you are using primary-target replication.

Related information

Failed Transaction (ATS and RIS) Files on page 199

Enabling ATS and RIS File Generation on page 200

Customizing the Replication Server Definition on page 99

Setting Up Failed Transaction Logging on page 105

Preparing the Database Server Environment

To prepare the database server environment, set database server environment variables and configuration parameters, and synchronize the operating system time on all participating database servers.

If you are using high-availability clusters with Enterprise Replication, set up your servers according to the instructions in Setting Up Database Server Groups for High-Availability Cluster Servers on page 89.

Setting Database Server Environment Variables

Certain environment variables must be set in a replication environment.

About this task

To configure the database server environment, verify that the following environment variables are set correctly:

- INFORMIXDIR is set to the full path of the HCL® Informix® directory.
- INFORMIXSERVER is set to the name of the default database server.
- INFORMIXSQLHOSTS is set to the full path to the SQLHOSTS file (UNIX™) or the SQLHOSTS registry host computer (Windows™).
- INFORMIXSQLHOSTS is set to the full path to the SQLHOSTS file.
- DELIMIDENT is not set or set to `n`. Enterprise Replication does not allow delimited identifiers.

Set configuration parameters for replication

You must set certain configuration parameters before you start Enterprise Replication. You can set other configuration parameters to customize the behavior of Enterprise Replication.

Parameters to set before you start replication

Set the following configuration parameters in the `onconfig` file on each database server that you want to include in the replication domain before you start replication:

- DBSERVERNAME specifies the name of the database server. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, set the DBSERVERNAME configuration parameter to the TCP connection and not to a shared-memory connection.
- CDR_DBSPACE specifies the dbspace for the **syscdr** database.

  If not set, the root dbspace is used.

  If not set and the database server has a storage pool with sufficient space, the dbspace is configured from the storage pool and the CDR_DBSPACE configuration parameter is set when you define a replication server.
- CDR_QUEUEMEM specifies the maximum amount of memory to be used for the send and receive queues.
- CDR_QDATA_SBSPACE specifies the location of the row data sbspace.

  If not set, you cannot define replication servers.

  If not set and the database server has a storage pool with sufficient space, the sbspace is configured from the storage pool and the CDR_QDATA_SBPACE configuration parameter is set when you define a replication server. Otherwise, you cannot define replication servers.
- CDR_SERIAL specifies how to generate non-overlapping (unique) values for serial columns across all database servers in the replication domain.
- CDR_TSINSTANCEID specifies how to generate unique identifiers for time series instances across all database servers in the replication domain.

Logging parameters

By default, if Enterprise Replication detects the potential for a log wrap situation when replication log processing lags behind the current log position, user transactions are blocked. You can configure Enterprise Replication to prevent the blocking of user transactions. Depending on the solutions you need, you might set the following configuration parameters in the `onconfig` file for each database server:

- CDR_LOG_LAG_ACTION specifies the actions that Enterprise Replication during a potential log wrap situation.
- LOG_STAGING_DIR specifies a directory in which compressed log files are staged.
- CDR_LOG_STAGING_MAXSIZE specifies the maximum size that Enterprise Replication can use to stage log files.
- CDR_MAX_DYNAMIC_LOGS specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.
- DYNAMIC_LOGS specifies that logical logs can be added dynamically.

Encryption parameters

If you want to encrypt network communications, set the following configuration parameters in the `onconfig` file for each database server:

- ENCRYPT_CDR specifies whether to enable encryption. The default value is `0`, which prevents encryption.
- ENCRYPT_CIPHERS specifies which ciphers and cipher modes are used for encryption.
- ENCRYPT_MAC controls the level of Message Authentication Code (MAC) used to ensure message integrity.

- ENCRYPT_MACFILE specifies the full path and file names of the MAC files.
- ENCRYPT_SWITCH specifies the number of minutes between automatic renegotiations of ciphers and keys. (The cipher is the encryption methodology. The secret key is the key that is used to build the encrypted data using the cipher.)

Other parameters

Set the following optional configuration parameters to customize your replication environment:

- CDR_DSLOCKWAIT specifies the number of seconds the data sync component waits for the database locks to be released. When replication is active on an instance, you can increase the amount of time to wait for lock resources to accommodate transactions on replicated tables.
- CDR_SUPPRESS_ATSRISWARN suppresses certain data sync error and warning codes from appearing in ATS and RIS files.
- CDR_DELAY_PURGE_DTC specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.
- GRIDCOPY_DIR specifies the default directory that is used by the ifx_grid_copy procedure.
- CDR_MAX_FLUSH_SIZE specifies the number of replicated transactions that are applied before the logs are flushed to disk.

Related reference

Enterprise Replication configuration parameter and environment variable reference on page 498

Related information

Managing Replication Servers on page 161

Row Data sbspaces on page 67

Serial data types and replication keys on page 29

DBSERVERNAME configuration parameter on page

DBSERVERALIASES configuration parameter on page

Adding a server to the domain by cloning a server on page 97

Configuring network encryption for replication servers on page 61

Time synchronization

Whenever you use replication that requires time stamp, time stamp with a stored procedure, or delete wins conflict resolution, you must synchronize the operating system times of the database servers that participate in the replicate.

All timestamps and internal computations are performed in Greenwich Mean Time (GMT) and have an accuracy of plus or minus one second.

Important: Enterprise Replication does not manage clock synchronization between database servers that participate in a replicate. You should use a product that supplies a network time protocol to ensure that times remain synchronized. For information on tools for synchronizing the times, refer to your operating system documentation.

To synchronize the time on one database server with the time on another database server, use one of the following commands, where *hostname* or *servername* is the name of the remote database server computer.

UNIX™

```
rdate hostname
```

Windows™

```
net time \\servername /set
```

```
net time /domain:servername /set
```

Important: These commands do not guarantee the times will remain synchronized. If the operating system times of the database servers do become out of sync or if their times move backward, time stamp or stored procedure conflict resolution might produce failures caused by incorrect time stamps.

Related information

Conflict Resolution on page 40

Delete wins conflict resolution rule on page 48

Time stamp conflict resolution rule on page 42

Adding a server to the domain by cloning a server on page 97

Preparing Data for Replication

The goal of data replication is to provide identical, or at least consistent, data on multiple database servers. This section describes how to prepare the information in your databases for replication.

When you define a new replicate on tables with existing data on different database servers, the data might not be consistent. Similarly, if you add a participant to an existing replicate, you must ensure that all the databases in the replicate have consistent values.

For more information, see Data Preparation Example on page 85.

Related information

Update-Anywhere Replication System on page 39

Preparing Consistent Data

In most cases, preparing consistent data simply requires that you decide which of your databases has the most accurate data and then that you copy that data onto the target database. If the target database already has data, for data consistency, you must remove that data before adding the copied data. For information on loading the data, see Load and unload data on page 82.

Blocking Replication

You might need to block replication so that you can put data into a database that you do not want replicated, perhaps for a new server or because you had to drop and re-create a table.

To block replication while you prepare a table, use the BEGIN WORK WITHOUT REPLICATION statement. This starts a transaction that does not replicate to other database servers.

The following code fragment shows how you might use this statement:

```
BEGIN WORK WITHOUT REPLICATION
LOCK TABLE office
DELETE FROM office WHERE description = 'portlandR_D'
COMMIT WORK
```

Related information

Load and unload data on page 82

Using DB-Access to Begin Work Without Replication

The following example shows how to use DB-Access to begin work without replication:

```
DATABASE adatabase;
BEGIN WORK WITHOUT REPLICATION
insert into mytable (col1, col2, ....)
   values (value1, value2, ....);
COMMIT WORK
```

Using ESQL/C to Begin Work Without Replication

The following example shows how to use Informix® ESQL/C to begin work without replication as well as update the Enterprise Replication shadow columns **cdrserver** and **cdrtime**:

```
MAIN (argc, argv)
   INT      argc;
   CHAR      *argv[];
{
   EXEC SQL CHAR      stmt[256];
   EXEC SQL database mydatabase;

   sprintf(stmt, "
BEGIN WORK WITHOUT REPLICATION"
);
   EXEC SQL execute immediate :stmt;
   EXEC SQL insert into mytable (col1, col2, ...)
      values (value1, value2, ...);
   EXEC SQL commit work;
}
```

Important: You must use the following syntax when you issue the BEGIN WORK WITHOUT REPLICATION statement from Informix® ESQL/C programs. Do not use the '$' syntax.

```
sprintf(stmt, "
BEGIN WORK WITHOUT REPLICATION"
);
EXEC SQL execute immediate :stmt;
```

Preparing to Replicate User-Defined Types

You must install and register user-defined types on all database servers prior to starting replication.

For Enterprise Replication to be able to replicate opaque user-defined types (UDTs), the UDT designer must provide two support functions, **streamwrite()** and **streamread()**. For more information, see Replication of opaque user-defined data types on page 33.

Preparing to Replicate User-Defined Routines

You must install and register user-defined routines on all database servers prior to starting replication.

Preparing Tables for Conflict Resolution

To use any conflict-resolution rule other than ignore or always-apply, you must define the shadow columns, **cdrserver** and **cdrtime** in the tables on both the source and target servers involved in replication.

To define the **cdrserver** and **cdrtime** shadow columns when you create a new table, use the WITH CRCOLS clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the two shadow columns:

```
CREATE TABLE customer(id int) WITH CRCOLS;
```

To add the **cdrserver** and **cdrtime** shadow columns to an existing replicated table:

1. Set alter mode on the table by running the **cdr alter --on** command.
2. Alter the table using the ADD CRCOLS clause.
3. Unset alter mode on the table by running the **cdr alter --off** command.

Adding CRCOLS columns to an existing table can result in a slow alter operation if any of the table columns have data types that require a slow alter. If a slow alter operation is necessary, make sure you have disk space at least twice the size of the original table, plus extra log space.

For example, the following statement adds the shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD CRCOLS;
```

You cannot drop conflict resolution shadow columns while replication is active. To drop the **cdrserver** and **cdrtime** shadow columns, stop replication and then use the DROP CRCOLS clause with the ALTER TABLE statement. For example, the following statement drops the two shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP CRCOLS;
```

Related information

Shadow columns on page 25

Shadow column disk space on page 65

Enterprise Replication shadow columns on page

Using the WITH CRCOLS Option on page

SQL statements and replication on page 21

Preparing Tables for a Consistency Check Index

To improve the speed of consistency checking with an index, you must define the **ifx_replcheck** shadow column in the tables on both the source and target servers involved in replication.

To define the **ifx_replcheck** shadow column when you create a new table, use the WITH REPLCHECK clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the **ifx_replcheck** shadow column:

```
CREATE TABLE customer(id int) WITH REPLCHECK;
```

To add the **ifx_replcheck** shadow column to an existing replicated table:

1. Set alter mode on the table by running the cdr alter --on command.
2. Alter the table using the ADD REPLCHECK clause.
3. Unset alter mode on the table by running the cdr alter --off command.

Because altering a table to add the **ifx_replcheck** shadow column is a slow alter operation, make sure you have disk space at least twice the size of the original table plus log space.

For example, the following statements add the **ifx_replcheck** shadow column to an existing table named **customer**:

```
ALTER TABLE customer ADD REPLCHECK;
```

To drop the **ifx_replcheck** shadow column, use the DROP REPLCHECK clause with the ALTER TABLE statement. For example, the following statements drop the **ifx_replcheck** shadow column from a table named **customer**:

```
ALTER TABLE customer DROP REPLCHECK;
```

For more information on the CREATE TABLE and ALTER TABLE statements, see the sections in the *HCL® Informix® Guide to SQL: Syntax*.

Related information

Shadow column disk space on page 65

Shadow columns on page 25

SQL statements and replication on page 21

Indexing the ifx_replcheck Column on page 183

Enterprise Replication shadow columns on page

Using the WITH REPLCHECK Keywords on page

Preparing tables without primary keys

The data columns in your table might not need a primary key. To replicate tables that do not have primary keys, you can specify a unique index or add the ERKEY shadow columns.

About this task

You can specify an existing unique index or unique constraint as the replication key when you define the replicate. Use the --key or --anyUniqueKey option with the cdr define replicate or cdr define template commands.

If you create a replicated table through a grid, the ERKEY shadow columns are automatically created and included in the replicate definition.

To add ERKEY shadow columns:

1. Add the ERKEY shadow columns when you create at table by using the WITH ERKEY keywords with the CREATE TABLE statement.

   Example

   For example, the following statement adds the ERKEY shadow columns to a table named **customer**:

   ```
   CREATE TABLE customer (id int) WITH ERKEY;
   ```

   Result

   The ERKEY shadow columns are named **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**.
2. Define the replicate. If you define a replicate by using the cdr define replicate command, include the --erkey option. If you define a template by using the cdr define template command, the ERKEY columns are included in the replicate definition automatically.

What to do next

To add the ERKEY shadow columns to an existing table that you want to start replicating:

1. Run the ALTER TABLE statement with the ADD ERKEY clause. For example, the following statement adds the ERKEY shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD ERKEY;
```

Occasionally, you might need to drop the ERKEY shadow columns; for example, if you are reverting to an earlier version of the database server.

To drop the ERKEY shadow columns from a replicated table:

1. Run the cdr remaster command without the --erkey option.
2. Run the DROP ERKEY clause with the ALTER TABLE statement.

For example, the following statement drops the ERKEY shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP ERKEY;
```

Related reference

cdr define replicate on page 342

cdr remaster on page 428

cdr change replicate on page 299

cdr define template on page 367

Related information

Unique key for replication on page 25

Creating replicated tables through a grid on page 136

Attaching a New Fragment to a Replicated Table on page 194

Using the WITH ERKEY Keywords on page

Enterprise Replication shadow columns on page

SQL statements and replication on page 21

Preparing Logging Databases

Databases on all server instances involved in replication must be created with logging. For best results, use unbuffered logging. For more information, see Unbuffered Logging on page 23.

Related reference

cdr start sec2er on page 450

Preparing for Role Separation (UNIX™)

You can use role separation to allow members of the DBSA group to run Enterprise Replication commands, in addition to the user **informix**. For some Enterprise Replication commands, you must grant the DBSA user additional permissions on tables or files. For non-root servers, role separation is not supported. Only the owner of a non-root server is allowed to run the Enterprise Replication commands that require additional permissions for a DBSA.

The DBSA user who runs Enterprise Replication commands must be a member of the DBSA group on all of the replication servers in the domain.

The following table describes the permissions that are needed for each command.

Table 12. Permissions for the DBSA user

Command	Type of Permission	Tables, Files, or Database

cdr check replicate
cdr check replicateset
cdr define replicate
cdr define replicateset
cdr define template
cdr realize template
cdr sync replicate
cdr sync replicateset
	INSERT
	UPDATE
	DELETE
		The tables that participate in replication. Must be granted on all replication servers in the domain.

The following commands with the --background option: • cdr check replicate • cdr check replicateset • cdr sync replicate • cdr sync replicateset	CONNECT or INSERT, depending on the object	sysadmin database: CONNECT ph_task table in the sysadmin database: INSERT Must be granted on the database server from which the command is run.
cdr define repair cdr start repair cdr stop repair cdr delete repair The following commands with the --syncdatasource option:	INSERT, UPDATE, or DELETE, depending on the table	The following syscdr tables: • rsncjobdef_tab: INSERT, UPDATE, DELETE • rsncjobdef: UPDATE • rsncprocnames_tab: INSERT • rsncjobdeps: INSERT Must be granted on all replication servers in the domain.

Table 12. Permissions for the DBSA user (continued)

Command	Type of Permission	Tables, Files, or Database
• cdr realize template • cdr start replicate • cdr start replicateset		
cdr repair cdr view atsdir cdr view risdir	read	ATS and RIS files Must be granted on the database server on which the files are located.

To update the permissions on a table or database, use the GRANT statement. For example, the following statement grants INSERT and UPDATE permissions on the rsncjobdef_tab table to the DBSA member with the user name of carlo:

```
GRANT INSERT, UPDATE ON rsncjobdef_tab TO carlo;
```

For more information about the GRANT statement, see the *HCL® Informix® Guide to SQL: Syntax*.

To update the permissions on ATS and RIS files, use an operating system command, such as the chown UNIX™ command.

Related reference

cdr check replicate on page 313

cdr check replicateset on page 325

cdr sync replicate on page 476

cdr sync replicateset on page 481

cdr repair on page 435

cdr view on page 486

cdr realize template on page 423

cdr define replicate on page 342

cdr define replicateset on page 355

cdr start replicate on page 445

cdr start replicateset on page 448

cdr define template on page 367

Load and unload data

You can load data into or unload data out of tables in your replication environment in various ways, depending on your circumstances.

If you have not yet set up your replication environment, for loading data, you can use the following tools:

- High-Performance Loader
- onunload and onload Utilities
- dbexport and dbimport utilities
- UNLOAD and LOAD statements
- External tables

When you unload and load data, you must use the same type of utility for both the unload and load operations. For example, you cannot unload data with the **onunload** utility and then load the data with a LOAD statement.

Existing replication environment

If you are adding a table to your already existing replication environment, Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication. You can synchronize the new table with data on the source server you specify when you start the new replicate, or when you add a new participant to an existing replicate. You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

If you want to use load and unload tools on tables that are already being replicated, you should block replication while you prepare the table. Unlogged changes to a table, such as data added by a light append, can be replicated to other tables.

If a table that you plan to replicate includes the CRCOLS or REPLCHECK shadow columns, the statements that you use for unloading the data must explicitly name the shadow columns. If you use the SELECT statement with * FROM *table_name* to the data to unload, the data from the shadow columns is not unloaded. To include the shadow columns in the unloaded data, explicitly name them. For example, use a statement like the following:

```
SELECT cdrserver, cdrtime, ifx_replcheck, * FROM table_name
```

If a table that you plan to replicate includes ERKEY shadow columns, you cannot unload and then load the data from these columns and preserve the original values. If you need to preserve the values of the ERKEY shadow columns, use synchronization to propagate the values.

Related information

Moving data with external tables on page

Blocking Replication on page 76

Setting Up Database Server Groups for High-Availability Cluster Servers on page 89

Initially Synchronizing Data Among Database Servers on page 120

Shadow columns on page 25

High-Performance Loader

The High-Performance Loader (HPL) provides a high-speed tool for moving data between databases.

How you use the HPL depends on how you defined the tables to replicate.

If the table contains shadow columns, you must:

- Include all the shadow column names in your map when you load the data.
- Use express mode to load data that contains shadow columns. You must perform a level-0 archive after completion.

You can also use deluxe mode without replication to load data. After a deluxe mode load, you do not need to perform a level-0 archive. Deluxe mode also allows you to load TEXT and BYTE data and opaque user-defined types.

For information about HPL, refer to the *IBM® Informix® High-Performance Loader User's Guide*.

onunload and onload Utilities

You can use the onunload and onload utilities to unload and load an entire table.

If you want to unload selected columns of a table, you must use either the UNLOAD statement or the HPL.

Restriction: You can only use the onunload and onload utilities in identical (homogeneous) environments.

If you use the onload utility while replication is active, you must synchronize the data after you finish loading the data.

Related information

The onunload and onload utilities on page

dbexport and dbimport Utilities

If you need to copy an entire database for replication, you can use the **dbexport** and **dbimport** utilities. These utilities unload an entire database, including its schema, and then re-create the database. If you want to move selected tables or selected columns of a table, you must use some other utility.

Related information

The dbexport and dbimport utilities on page

UNLOAD and LOAD Statements

The UNLOAD and LOAD statements allow you to move data within the context of an SQL program.

If the table contains shadow columns, you must:

- Include all shadow columns in your map when you unload the data.
- List the columns that you want to load in the INSERT statement and explicitly include the shadow columns in the list when you load your data.

For more information about the UNLOAD and LOAD statements, see the *HCL® Informix® Guide to SQL: Syntax*.

Data Preparation Example

The following examples show how to add a new participant (**delta**) to an existing replicate by two different methods:

- Using the **cdr start replicate** command

  This method is simple and can be done while replication is online.

- Using the LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION statements.

  If you use HPL, this method can be faster for a large table.

Replicate **zebra** replicates data from table **table1** for the following database servers: **alpha**, **beta**, and **gamma**.

The servers **alpha**, **beta**, and **gamma** belong to the server groups **g_alpha**, **g_beta**, and **g_gamma**, respectively. Assume that **alpha** is the database server from which you want to get the initial copy of the data.

Using the cdr start replicate Command

About this task

To add a new participant to an existing replicate

1. Declare server **delta** to Enterprise Replication.
   For example:

   ```
   cdr def ser -c delta -I -S g_alpha g_delta
   ```

   At the end of this step, all servers in the replication environment include information in the **syscdr** database about **delta**, and **delta** has information about all other servers.

2. Add **delta** as a participant to replicate **zebra**.
   For example:

   ```
   cdr cha rep -a zebra "dbname@g_delta:owner.table1"
   ```

   This step updates the replication catalog. The servers **alpha**, **beta**, and **gamma** do not queue any qualifying replication data for **delta** because the replicate on **delta**, although defined, has not been started.

3. Start replication for replicate **zebra** on **delta**.

   ```
   cdr sta rep zebra g_delta -S g_alpha -e delete
   ```

   The **-S g_alpha** option specifies that the server **alpha** be used as the source for data synchronization.

   The **-e delete** option indicates that if there are rows on the target server, **delta**, that are not present on the synchronization data server (**alpha**) then those rows are deleted

   Do not run any transactions on **delta** that affect table **table1** until you finish the synchronization process.

Using LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION

When you add a new participant to an existing replicate, you can unload and load data without replication.

About this task

To add a new participant to an existing replicate

1. Add the server **delta** to the Enterprise Replication domain.
   For example:

   ```
   cdr def ser -c delta -I -S g_alpha g_delta
   ```

   At the end of this step, all servers in the replication environment include information in the **syscdr** database about **delta**, and **delta** has information about all other servers.
2. Add **delta** as a participant to replicate **zebra**.
   For example:

   ```
   cdr cha rep -a zebra "P dbname@g_delta:owner.table1" \
   "select * from table1"
   ```

   This step updates the replication catalog. The servers **alpha**, **beta**, and **gamma** do not queue any qualifying replication data for **delta** because the replicate on **delta**, although defined, has not been started.
3. Suspend server **delta** on **alpha**, **beta**, and **gamma**.

   ```
   cdr sus ser g_delta g_alpha g_beta g_gamma
   ```

   As a result of this step, replication data is queued for **delta**, but no data is delivered.
4. Start replication for replicate **zebra** on **delta**.

   ```
   cdr sta rep zebra g_delta
   ```

   This step causes servers **alpha**, **beta**, and **gamma** to start queuing data for **delta**. No data is delivered to **delta** because **delta** is suspended. Then, **delta** queues and delivers qualifying data (if any) to the other servers.

   Do not run any transactions on **delta** that affect table **table1** until you finish the synchronization process.
5. Unload data from table **table1** using the UNLOAD statement or the **unload** utility on HPL.
6. Copy the unloaded data to **delta**.
7. Start transactions with BEGIN WORK WITHOUT REPLICATION, load the data using the LOAD statement, and commit the transactions.
   If you used the HPL to unload the data in step 5 on page 86, then use the HPL Deluxe load without replication to load the data into **table1** on **delta**.
8. Resume server **delta** on **alpha**, **beta**, and **gamma**.
   Example

   ```
   cdr res ser g_delta g_alpha g_beta g_gamma
   ```

This step starts the flow of data from **alpha**, **beta**, and **gamma** to **delta**.

At this point, you might see some transactions aborted because of conflict. Transactions can abort because **alpha**, **beta**, and **gamma** started queuing data from **delta** in step 4. However, those same transactions might have been moved in steps 5 and 7.

Results

You must declare replication on server **delta** and then immediately suspend replication because, while you are preparing the replicates and unloading and loading files, the other servers in the replicate (**alpha**, **beta**, and **gamma**) might be collecting information that needs to be replicated. After you finish loading the initial data to **delta** and resume replication, the information that was generated during the loading process can be replicated.

Using High-Availability Clusters with Enterprise Replication

In This Chapter

This chapter covers how to include other data replication solutions, such as high-availability data replication, in your Enterprise Replication system. The following topics are covered:

- The design of a high-availability cluster replication system
- Preparing a high-availability cluster database server
- Managing Enterprise Replication with a high-availability cluster

For a complete description of data replication, see the *HCL® Informix® Administrator's Guide*.

High-availability replication systems

You can combine HCL Informix® Enterprise Replication and high-availability clusters to create a high-availability replication system.

A high-availability cluster consists of two types of database servers:

- A primary database server, which receives updates, and can participate in Enterprise Replication.
- Secondary servers, which mirror the primary server and are perpetually applying logical-log records from the primary server, and cannot participate in Enterprise Replication.

A minimal high-availability cluster consists of a primary server and a HDR secondary server that are tightly coupled. Transactions on the primary server are not committed until the log records containing the transactions are sent to the HDR secondary server.

High-availability clusters can also contain shared-disk (SD) secondary servers and remote standalone (RS) secondary servers. A SD secondary server does not maintain a copy of the physical database on its own disk space; it shares disks with the primary server. An RS secondary servers maintains a copy of the physical database on its own disk space.

If the primary server in a high-availability cluster becomes unavailable, one of the secondary servers takes over the role of the primary server. In a high-availability replication system, if the primary server fails, a secondary database is promoted to primary server, and Enterprise Replication can continue with the new primary server.

You can configure Connection Managers to direct client requests to replication servers, and to control which secondary server takes over if the primary server becomes unavailable.

A high-availability replication system is effective when you use a hierarchical or a forest of trees topology.

Related information

Update-Anywhere Replication System on page 39

The sqlhosts information on page

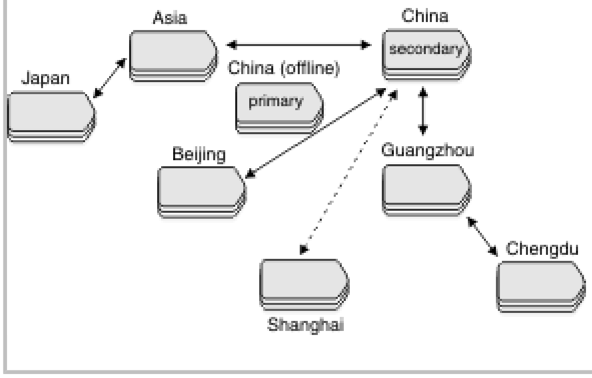
High-Availability Clusters in a Hierarchical Tree Topology

With a hierarchical tree topology, parent servers are good candidates for using high-availability clusters to provide backup servers.

The following example is based on the example in Figure 12: Hierarchical Tree Topology on page 53.

If **China** fails, then **Beijing** and **Shanghai** can no longer replicate with other servers in the replication system; **Guangzhou** and **Chengdu** can replicate only with each other. However, if **China** was part of a high-availability cluster, when it failed, the secondary server would replace it and replication would continue, as illustrated in Figure 14: Hierarchical Tree Topology with HDR on page 88.

Figure 14. Hierarchical Tree Topology with HDR



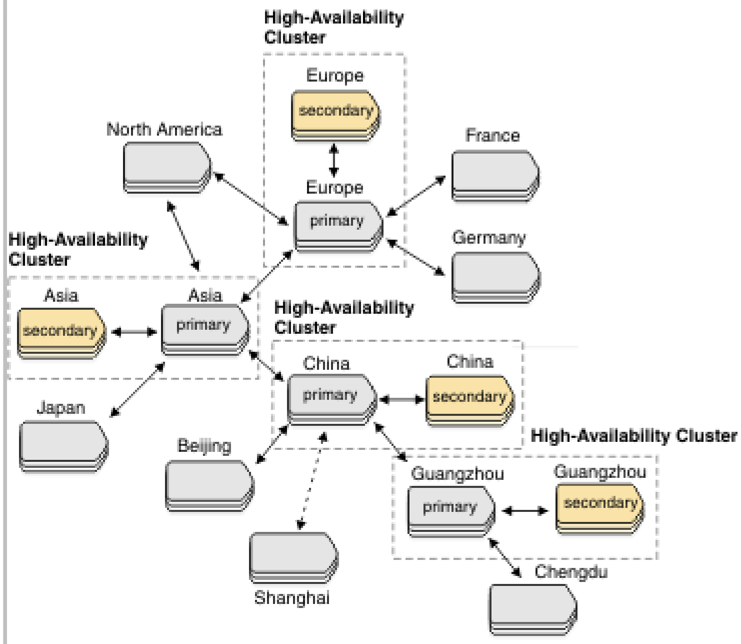
In this example, **Asia** and **Guangzhou**, which are also parent servers, might also benefit from using a high-availability cluster to ensure high availability.

Using high-availability clusters in a forest of trees topology

Use a high-availability cluster to ensure that all servers retain access to the replication system in a forest of trees topology.

For example, in Figure 13: Forest-of-Trees Topology on page 54, **Asia**, **Europe**, **China**, and **Guangzhou** should use high-availability clusters to provide backup servers, as illustrated in Figure 15: High-Availability Clusters in a Forest-of-Trees Topology on page 89.

Figure 15. High-Availability Clusters in a Forest-of-Trees Topology

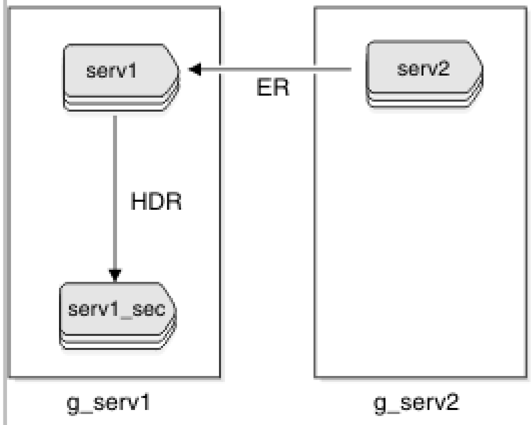


Setting Up Database Server Groups for High-Availability Cluster Servers

When defining a high-availability cluster within Enterprise Replication, the cluster must appear to be a single logical entity within the replication domain. Define the servers within the same database server group in the `sqlhosts` file.

For example, Figure 16: Database Server Groups for Enterprise Replication with HDR on page 89 illustrates two Enterprise Replication nodes, one of which is an HDR pair.

Figure 16. Database Server Groups for Enterprise Replication with HDR



In this example, the HDR pair consists of the primary server, **serv1**, and the secondary server, **serv1_sec**. These two servers belong to the same database server group, **g_serv1**. The non-HDR server, **serv2**, belongs to the database server group **g_serv2**. The following example displays the `sqlhosts` file for this configuration:

```
#dbservername    nettype    hostname     servicename  options
 g_serv1         group      -            -            i=1
 serv1           ontlitcp   machine1pri  port1        g=g_serv1
 serv1_sec       ontlitcp   machine1sec  port1        g=g_serv1
 g_serv2         group      -            -            i=2
 serv2           ontlitcp   machine2     port1        g=g_serv2
```

Important: If you use the g=server option in the group member definition, you can put the definition anywhere in the `sqlhosts` file.

Either HDR or Enterprise Replication can be set up first on the HDR pair **serv1** and **serv1_sec**, but Enterprise Replication **cdr** commands must be run only on the primary server. If any **cdr** commands are attempted on the secondary server, a –117 error is returned: `Attempting to process a cdr command on an HDR secondary server`.

Related information

Load and unload data on page 82

Creating sqlhost group entries for replication servers on page 58

sqlhosts connectivity information on page

Managing Enterprise Replication with High-Availability Clusters

This section describes how to manage Enterprise Replication with HDR in the following areas:

- Failure of the primary server in a high-availability cluster
- Performance considerations

Failover for High-availability clusters in an Enterprise Replication environment

About this task

If you configure connection management for failover, Connection Managers can promote a secondary server to the primary-server if the primary server fails. If connection management is not configured to control failover, the onmode -d make primary command can promote a secondary server to the primary-server role. In either of these cases, Enterprise Replication automatically connects to the new primary server.

If the primary server fails, and you manually change a secondary server to a standard server, you must complete the following steps to prevent Enterprise Replication from starting on all servers in cluster.

Run the following commands on the secondary server:

1. onmode -s
2. onmode -d standard
3. cdr start

If Enterprise Replication is running on the secondary server, and you want to restart the server that was the primary server, but without Enterprise Replication and high-availability cluster replication, run the oninit -D command. You can then stop Enterprise Replication on the standard server and reestablish the primary server.

First, run the following commands on the standard server:

1. cdr stop
2. onmode -d secondary *primary_ha_alias*

Second, run the following commands on the primary server:

1. oninit
2. cdr start

To split an active cluster into two standalone servers, you must restart the database servers with the oninit -D command to prevent Enterprise Replication from starting on either server after they are split.

To remove a server from a cluster, run the cdr delete server -force *ha_alias* command, where *ha_alias* is an Enterprise Replication group name, to remove Enterprise Replication from that server. For example, the two HDR servers are being split and the secondary server is to be used for reporting purposes. After the report processing is complete, HDR can be reestablished. cdr delete server on page 377 shows how to remove a secondary server from a high-availability cluster and Enterprise Replication.

Table 13. Removing the Secondary Server from a cluster and ER

Step	On the Primary	On the Secondary
1.	onmode -d standard *secondary_ha_alias*	
2.		Run onmode -d standard
3.		Run cdr delete server -f *ha_alias*

If the HDR primary server has problems communicating to its secondary server, Enterprise Replication is in a suspended state until one of the following actions is taken:

- Resolve the connection problem between HDR pairs.
- Convert the primary server to standard mode.

Related reference

cdr delete server on page 377

Related information

Connection management through the Connection Manager on page

Replication latency for secondary servers

When you combine Enterprise Replication with high-availability clusters, replication latency can increase.

When Enterprise Replication is running on a high-availability cluster, some operations cannot be performed until the logs are shipped to the secondary server. By default, the logs are shipped to secondary servers after 50 replicated transactions are applied, or 5 seconds elapse. This delay prevents possible inconsistency within the Enterprise Replication domain during a failover to a secondary server.

You can control replication latency for high-availability data replication (HDR) servers in one of the following ways

- Set HDR replication to fully synchronous, nearly synchronous, or asynchronous mode.
- Set HDR replication to HDR SYNC.
- Adjust the DRINTERVAL configuration parameter to specify a different interval between flushing the high-availability data-replication buffer.

If you combine Enterprise Replication with shared-disk secondary servers, you can reduce replication latency by setting the CDR_MAX_FLUSH_SIZE configuration parameter to 1 to flush the logs after each replicated transaction.

Related reference

CDR_MAX_FLUSH_SIZE configuration parameter on page 509

Related information

DRINTERVAL configuration parameter on page

HDR_TXN_SCOPE configuration parameter on page

Replication of primary-server data to secondary servers on page

Fully synchronous mode for HDR replication on page

Nearly synchronous mode for HDR replication on page

Defining Replication Servers, Replicates, Participants, and Replicate Sets

These topics describe the steps defining and starting Enterprise Replication.

About this task

To define and start replication:

1. Initialize the database server.
2. Create a replication domain by defining replication servers.
3. Configure replication by defining replicates, and optionally grouping replicates into a replicate set.

The replicate definition includes information about the participants, replication options, frequency, and conflict-resolution rules and scope.

4. Specify the data to replicate by defining participants.
   A participant definition specifies the data (database, table, and columns) that should be replicated.
5. Synchronize the data among the replicates.

Starting Database Servers

The database server must be online before you can define it as a replication server.

About this task

To bring the server from offline to online, issue the following command for your operating system.

Operating System	Command
UNIX™	**oninit**
Windows™	**start** *dbservername*

To bring the server from quiescent mode to online on either UNIX™ or Windows™, enter **onmode -m**.

For more information on initializing the database server, see the chapter on database server operating modes in the *HCL® Informix® Administrator's Guide*.

Defining Replication Servers

You must define a replication server to create a replication domain or to add a server to an existing domain.

Before you begin

The database server must be online.

The CDR_QDATA_SBSPACE configuration parameter must be set to a valid value.

You must be the Enterprise Replication server administrator to define the replication server.

You can define replication servers using two different methods:

- The cdr utility
- Cloning

About this task

To define the replication server in a new domain by using the cdr utility, use the **cdr define server** command to connect to the database server and specify the database server group name. For example, the following command connects to a server called **stan** and creates a domain containing the database server group **g_stan**:

```
cdr define server  --connect=stan --init g_stan
```

The **--init** option specifies the database server group to add to the replication domain. If the INFORMIXSERVER environment variable is not set to the server that you are defining, specify the **--connect=***server_name* option. You can also configure replication attributes for the server.

To define a replication server in an existing domain by using the cdr utility, include the **--sync=***sync_server* option with the **cdr define server** command to synchronize the global catalog with an existing server. For example, the following command adds a server group named **g_oliver** to the domain created in the previous command, using **g_stan** as the synchronization server:

```
cdr define server  --connect=oliver --init g_oliver --sync=g_stan
```

You can specify any existing server in the domain, however, if you define a server as a nonroot or a leaf server, then the synchronization server becomes the parent of the new server. For example, if you add a server **kauai** as a leaf server and want its parent to be **hawaii**, then specify **hawaii** with the **--sync** option.

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 and the server has a storage pool with sufficient space, the cdr define server command automatically creates the necessary disk space and sets the configuration parameters to appropriate values.

The maximum number of replication servers that you can define is 32767.

Related reference

cdr define server on page 357

cdr define replicate on page 342

Related information

Row Data sbspaces on page 67

Choosing a Replication Network Topology on page 51

Enterprise Replication Server administrator on page 18

Setting Up Failed Transaction Logging on page 105

Modify server attributes on page 161

Creating a new domain by cloning a server

You can create a new replication domain by cloning a server and then converting the two Informix® database servers to replication servers. Use cloning and conversion if you want to set up replication based on the data on a source server that is not yet running Enterprise Replication.

About this task

Because the source server does not have Enterprise Replication defined, you use the ifxclone utility to create a cluster containing a primary server and remote stand-alone (RS) secondary server. The conversion process converts the cluster to a pair of replication servers in a new domain.

To create a new domain with two replication servers:

1. On the source server, prepare the server environment for Enterprise Replication, such as configuring `sqlhosts` information and setting the necessary configuration parameters.
2. On both servers, complete the ifxclone prerequisites for all servers, such as setting the required configuration parameters and environment variables.
3. On the target server, complete the ifxclone prerequisites for an RS secondary server, such as creating all of the chunks that exist on the source server. You can use the --createchunkfile option (-k) of the ifxclone utility to automatically create cooked chunks on the target server.
4. On the target server, run the ifxclone command with the --disposition=RSS option to clone the data and the configuration of the source server onto the target server. Do not include the --useLocal option.
5. On the source server, run the cdr check sec2er command to determine if conversion to replication servers is possible.
6. Solve any error conditions identified by the cdr check sec2er command and rerun it until its output indicates that conversion will be successful. You can also solve warning conditions.
7. On the source server, run the cdr start sec2er command to convert both servers to replication servers and create a new replication domain.

What to do next

To add other servers to the domain, you can clone a replication server.

Related information

The ifxclone utility on page

Preparing the Replication Environment on page 55

Adding a server to the domain by cloning a server on page 97

Example of creating a new replication domain by cloning

This is an example of creating a new replication domain based on the data and configuration on a source database server that does not have replication defined. The three additional replication servers in the domain are added by cloning the source server.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix® database server installed. The source server contains the **stores_demo** database.

1. On the **serv1** server, set the CDR_QDATA_SBSPACE configuration parameter.
2. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_CLONE configuration parameter to `1` in the `onconfig` file.
3. On the **serv1** server, add the following `sqlhosts` information about **serv1** and **serv2**:

```
gserv1     group    -                      -     i=143
serv1      ontlitcp ny.usa.com             1230  g=gserv1
gserv2     group    -                      -     i=144
serv2      ontlitcp tokyo.japan.com        1231  g=gserv2
```

4. On both the **serv1** and **serv2** servers, complete the ifxclone prerequisites for all servers, such as setting the required configuration parameters and environment variables.

Set these environment variables:

- **INFORMIXDIR**
- **INFORMIXSERVER**
- **INFORMIXSQLHOSTS**
- **ONCONFIG**

Set these configuration parameters to the same values on both servers:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT
- LOGBUFF
- LOGFILES
- LOGSIZE
- LTAPEBLK
- LTAPESIZE
- ROOTNAME
- ROOTSIZE
- PHYSBUFF
- PHYSFILE
- STACKSIZE
- TAPEBLK
- TAPESIZE

5. On the **serv2** server, create all of the chunks that exist on the **serv1** server. You can use the --createchunkfile option (-k) of the ifxclone utility to automatically create cooked chunks on the target server.
6. On the **serv2** server, run the ifxclone command with the --disposition=RSS option to clone the data and the configuration of the **serv1** server onto the **serv2** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv2 --targetIP=192.168.0.2
--targetPort=1231 --disposition=RSS --createchunkfile
```

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv2 --targetIP=192.168.0.2
--targetPort=1231 --disposition=RSS
```

7. On the **serv1** server, run the cdr check sec2er command to determine if conversion to replication servers is possible:

```
$cdr check sec2er -c gserv1 gserv2
Secondary conversion to ER is possible.
```

8. On the **serv1** server, run the cdr start sec2er command to convert both servers to replication servers, create a new replication domain, create and start replicates based on all the tables on the **serv1** server:

```
cdr start sec2er -c gserv1 gserv2
```

9. On the **serv3** and **serv4** servers, provision chunk paths and other storage to the same paths and at least the same sizes as on the **serv1** server.

10. On the **serv3** server, run the ifxclone command with the --disposition=ER option to clone the data and the configuration of the **serv1** server onto the **serv3** server:

    ```
    ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
    --sourcePort=1230 --target=serv3 --targetIP=192.168.0.3
    --targetPort=1232 --disposition=ER
    ```

11. On the **serv4** server, run the ifxclone command with the --disposition=ER option to clone the data and the configuration of the **serv1** server onto the **serv4** server:

    ```
    ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
    --sourcePort=1230 --target=serv4 --targetIP=192.168.0.4
    --targetPort=1233 --disposition=ER
    ```

12. Edit the `sqlhosts` files on all four servers so that they each have the following information:

    ```
    gserv1     group    -                    -     i=143
    serv1      ontlitcp ny.usa.com           1230  g=gserv1
    gserv2     group    -                    -     i=144
    serv2      ontlitcp tokyo.japan.com      1231  g=gserv2
    gserv3     group    -                    -     i=145
    serv3      ontlitcp rome.italy.com       1232  g=gserv3
    gserv4     group    -                    -     i=146
    serv4      ontlitcp perth.australia.com  1233  g=gserv4
    ```

Related reference

cdr start sec2er on page 450

cdr check sec2er on page 334

Related information

The ifxclone utility on page

Adding a server to the domain by cloning a server

You can add a replication server to an existing replication domain by using the ifxclone utility to clone an existing replication server onto a target database server.

Before you begin

Enterprise Replication must be active on the source server. The source server should not have any stopped or suspended replicates or any shadow replicates defined.

You must be user **informix** or member of the **informix** group to run the ifxclone utility.

HCL Informix® database software must be installed on the target server.

About this task

Cloning a server defines replication on the target server, copies the data, and adds the target server to all replicates in which the source server participates. The `onconfig` file and the sqlhosts file are copied from the source server to the target server and updated with the target server information.

To clone a replication server by using the ifxclone utility:

1. On the source server, set the value of the ENABLE_SNAPSHOT_COPY configuration parameter to `1` in the `onconfig` file.
2. On the target server, create the following directories, if they exist on the source server. The directories must be the same on both servers:

   Choose from:

   - ATS and RIS directories
   - Log staging directory
3. On the target server, synchronize the system clock with the source server.
4. On the target server, provision chunk paths and other storage to the same paths and at least the same sizes as on the source server. Ensure that the target server has at least as much memory and disk space resources as the source server. You can use the --createchunkfile option (-k) of the ifxclone utility to automatically create cooked chunks on the target server.
5. On the target server, run the ifxclone command.

   You must provide the following information to the ifxclone utility:

   - Source server name
   - Source IP address
   - Source port
   - Target server name
   - Target IP address
   - Target port

   Include the --disposition=ER option.

   Optional: Include the --createchunkfile option.

   If the source server replicates serial columns, use the --configParam option to set the value of the CDR_SERIAL configuration parameter to ensure that serial values do not conflict between replication servers.

   Example

   The ifxclone utility has the following format for cloning a replication server:

   ```
   ifxclone --source=source_name --sourceIP=source_IP
   --sourcePort=source_port --target=target_name
    --targetIP=target_IP --targetPort=target_port
   --disposition=ER --createchunkfile
   ```

   ```
   ifxclone --source=source_name --sourceIP=source_IP
   --sourcePort=source_port --target=target_name
    --targetIP=target_IP --targetPort=target_port
   --disposition=ER
   ```
6. On all other replication servers in the domain, edit the sqlhosts file to add entries for the new replication server.

Related reference

Set configuration parameters for replication on page 72

Related information

The ifxclone utility on page

Time synchronization on page 74

Creating a new domain by cloning a server on page 94

Adding a replication server to a grid by cloning on page 132

Customizing the Replication Server Definition

You can specify replication attributes of a server when you define it.

About this task

When you define a replication server, you can specify the following attributes in the **cdr define server** command:

- Set the idle timeout.

  To specify the time (in minutes) that you want to allow the connection between two Enterprise Replication servers to remain idle before disconnecting, use the **--idle=***timeout* option.

  You can later change the values of this attribute with the **cdr modify server** command.

- Specify the location of the ATS and RIS directories.

  To use ATS, specify the directory for the Aborted Transaction Spooling (ATS) files for the server using **--ats=***dir* or**--ris=***dir* . To prevent either ATS or RIS file generation, set the directory to **/dev/null** (UNIX™) or **NUL** (Windows™).

  You can later change the values of these attributes with the **cdr modify server** command.

- Specify the format of the ATS and RIS files.

  Use the **-atsrisformat=***type* option to specify whether the ATS and RIS files are generated in text format, XML format, or both formats.

  You can later change the values of this attribute with the **cdr modify server** command.

- Specify the type of server if you are using hierarchical replication:
  - To specify the server as a nonroot server, use the **--nonroot** option.
  - To specify the server as a leaf server, use the **--leaf** option.

  If neither **--leaf** nor **--nonroot** is specified, the server is defined as a root server. The parent server is the server specified by the **--sync=***sync_server* option.

Related reference

cdr define server on page 357

cdr modify server on page 421

Related information

Choosing a Replication Network Topology on page 51

Enterprise Replication Terminology on page 3

Creating ATS and RIS directories on page 71

Define a replicate

To define a replicate, use the cdr define replicate command.

You can provide the following information in the replicate definition:

- Participants
- Create as a master replicate
- Conflict resolution rules and scope
- Replication frequency
- Error logging
- Replicate full rows or only changed columns
- IEEE or canonical message formats
- Database triggers
- Code set conversion between replicates
- Replication key
- Serial or parallel processing

After you define the replicate and participants, you must manually start the replicate by running the cdr start replicate command.

The maximum number of replicates that you can define as participants on a particular replication server is 32767.

Participant definitions

You must define a participant for each server that is involved in the replicate definition by running the cdr define replicate command. Each participant in a replicate must specify a different database server.

Each participant definition includes the following information:

- Database server group name
- Database in which the table to be replicated resides
- Table name
- Table owner
- Participant type

For a primary-target replication system, you can specify the participant type as primary, receive-only, or send-only.For a primary-target replication system, you can specify the participant type as primary or receive-only. If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere, by default.

- SELECT statement and optional WHERE clause

Restriction: Do not create more than one participant definition for each row and column to replicate. If the participant is the same, Enterprise Replication attempts to insert or update duplicate values during replication. For example, if one participant modifier includes WHERE x < 50 and another includes WHERE x < 100, Enterprise Replication sends the data for when x is between 50 and 100 twice.

Related reference

Participant and participant modifier on page 257

Related information

Primary-Target Replication System on page 35

Defining Replicates on Table Hierarchies

When you define replicates on inherited table hierarchies, use the following guidelines to replicate operations:

- For both the parent and child tables, define a replicate on each table.
- For only the parent table (not the child table), define a replicate on the parent table only.
- For only the child table (not the parent table), define a replicate on the child table only.

Replicate types

You can choose a replicate type depending on whether you want the schema definitions on all participants to be the same. A master replicate enforces consistency between the schema definitions of the participants and the schema definition on a designated server. A classic replicate does not check the schema definitions of the participants.

To guarantee consistency between the nodes in your Enterprise Replication environment, define master replicates by running the cdr define replicate command with the --master option. By default, replicates are created as classic replicates. By default, replicates are master replicates. If you do not specify a master server, the master replicate is based on the first participant. Dictionary information is then stored about replicated column attributes for the participant you specify. Enterprise Replication checks for consistency between the master definition and local participant definitions. Checks are run when the replicate is defined and each time a new participant is added to the replicate, thus avoiding runtime errors. Verification also occurs each time that the master replicate is started on a server.

If you do not want to verify the schema, create a classic replicate. For example, if you want to create a data consolidation system in which one server only receives data from other servers that only send data, create a classic replicate.

Defining a replicate as a master replicate provides several advantages:

- Ensures data integrity by verifying that all participants in the replicate have table and replicated column attributes that match the master replicate definition.
- Provides automatic table generation on participants that do not already contain the table that is specified in the master replicate. However, Enterprise Replication cannot create tables with user-defined data types.
- Allows alter operations on the replicated tables.

When you define a master replicate, you must specify a participant that is on the server for which you are running the command. This participant is used to create the dictionary information for the master replicate. When you define a master replicate, you can specify a participant that is on the server for which you are running the command. By default, the first participant that you list in the cdr define replicate command is the used to create the dictionary information for the master replicate. The additional participants in the cdr define replicate command are verified against the master definition and added to the replicate if they pass validation. If any participant fails validation, the cdr define replicate command fails and that participant is disabled.

Related reference

cdr define template on page 367

cdr define replicate on page 342

Master Replicate Verification

About this task

Enterprise Replication verifies the following information about a participant when the participant is added to the master replicate:

- The participant contains all replicated columns.
- The replicated columns in the participant have the correct data types. For columns that are user-defined data types, only the names of the data types are verified.
- Optionally, the replicated columns in the participant have the same column names as the master replicate.

Creating Strict Master Replicates

About this task

You can create a strict master replicate in which all participants have the same replicated column names by using the **--name=y** option. This option specifies that when the master replicate verification is done for a new participant, that the column names on the participant must be identical to the column names of the master replicate. Strict master replicates allow you to perform the following tasks:

- Alter operations on replicated tables. For more information, see Alter, rename, or truncate operations during replication on page 188.
- Remastering by using the **cdr remaster** command. For more information, see Remastering a Replicate on page 195.

You can modify an existing master replicate to remove name verification by using the **--name=n** option of the **cdr modify replicate** command.

Related reference

cdr modify replicate on page 415

Creating Empty Master Replicates

About this task

You can create an empty master replicate by using the **--empty** option. This option allows you to specify a participant as the basis of the master replicate but not include that participant in the replicate. Creating an empty replicate can be convenient in large environments in which you later add all participants using scripts.

When you define an empty master replicate, you must specify only one participant in the **cdr define replicate** command. This participant is used to create the master dictionary information but is not added to the replicate.

The **--empty** option is only supported for master replicates, you cannot use it without the **--master** option.

Defining Shadow Replicates

About this task

A *shadow replicate* is a copy of an existing, or primary, replicate. Enterprise Replication uses shadow replicates to manage alter and repair operations on replicated tables. You must create a shadow replicate to perform a manual remastering of a replicate that was defined with the **-n** option. See Resynchronize data manually on page 188 for information about how you can repair, or remaster, your replicated data. After creating the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate using the **cdr swap shadow** command.

You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option, as described in cdr define replicate on page 342.

When you define a shadow replicate, its state is always set to the same state as the primary replicate. If you change the state of the primary replicate, all its shadow replicates' states are also changed to the same state.

You cannot delete a primary replicate if it has any shadow replicates defined. You must first delete the shadow replicates, and then the primary replicate.

You cannot modify a primary replicate (using the **cdr modify replicate** command) if it has any shadow replicates defined. Also, you cannot modify shadow replicates directly.

You cannot suspend or resume a primary replicate (using the **cdr suspend replicate** or **cdr resume replicate** command) if it has any shadow replicates defined. Also, you cannot suspend or resume shadow replicates directly. If the primary replicate and its shadow replicates are part of an exclusive replicate set, you can suspend or resume the entire replicate set using the **cdr suspend replicate** or **cdr resume replicate** command.

You cannot add a participant to a shadow replicate:

- If the participant is not part of the primary replicate's definition
- After remastering the replicate

If the primary replicate is part of an exclusive replicate set, any shadow replicates you define are automatically added to that replicate set.

If you add a primary replicate to an exclusive replicate set, all its shadow replicates are also automatically added. If you delete a primary replicate from an exclusive replicate set, all its shadow replicates are also automatically deleted.

Specifying Conflict Resolution Rules and Scope

You specify the conflict resolution rule in the replicate definition.

About this task

For update-anywhere replication systems, you must specify the conflict-resolution rules in the replicate definition using the **--conflict=***rule* option to the **cdr define replicate** command. The conflict resolution rule option names are:

- **always**
- **deletewins**
- **ignore**
- **timestamp**
- ***routine_name***

  If you use an SPL routine for your conflict-resolution rule, you can also use the **--optimize** option to specify that the routine is optimized.

You can also specify the scope using the **--scope=*scope*** option:

- **transaction** (default)
- **row**

Related reference

cdr define replicate on page 342

Related information

Update-Anywhere Replication System on page 39

Conflict resolution rule on page 40

Conflict Resolution Scope on page 50

Specifying Replication Frequency

About this task

The replication frequency options allow you to specify the interval between replications, or the time of day when an action should occur. If you do not specify the frequency, the default action is that replication always occurs immediately when data arrives.

The frequency options are:

- **--immed**
- **--every=*interval***
- **--at=*time***

For more information, see Frequency Options on page 287.

Important: If you use time-based replication and two tables have referential constraints, the replicates must belong to the same exclusive replicate set. For more information, see Exclusive Replicate Sets on page 118.

Setting Up Failed Transaction Logging

The Aborted Transaction Spooling (ATS) files and Row Information Spooling (RIS) files contain information about failed transactions and aborted rows. You can use this information to help you diagnose problems that arise during replication.

About this task

To configure your replicate to use ATS and RIS

1. Set up the ATS and RIS directories.
2. Specify the location of the ATS and RIS directories when you define your server.
3. Specify that the replicate use ATS and RIS when you define the replicate by including the **--ats** and **--ris** options in the replicate definition.

Results

Tip: Until you become thoroughly familiar with the behavior of the replication system, select both ATS and RIS options.

Related reference

Replicate only changed columns on page 106

cdr define replicate on page 342

Related information

Creating ATS and RIS directories on page 71

Defining Replication Servers on page 93

Monitor and troubleshooting Enterprise Replication on page 197

Replicate only changed columns

You can choose to replicate only those columns that have changes instead of entire rows.

By default, even if only one column changes, Enterprise Replication replicates the entire row, except columns that contain unchanged large objects.

You can change the default behavior to replicate only the columns that changed. To replicate only changed columns, include the --fullrow=n option in the replicate definition. Enterprise Replication always sends the replication key columns, even if you specify to replicate only changed columns.

Replicating only the columns that changed has the following advantages:

- Sends less data, because only the modified data is sent
- Uses less Enterprise Replication resources, such as memory

If Enterprise Replication replicates an entire row from the source, and the corresponding row does not exist on the target, Enterprise Replication applies the update as an insert, also known as an *upsert*, on the target (unless you are using the delete wins conflict resolution rule). By replicating the entire row, Enterprise Replication corrects any errors during replication. If any errors occur in an update of the target database server (for example, a large object is deleted before Enterprise Replication can send the data), the next update from the source database server (a complete row image) corrects the data on the target server.

Replicating only the columns that changed has the following disadvantages:

- Enterprise Replication does not apply upserts.

  If the row to replicate does not exist on the target, Enterprise Replication does not apply it. If you set up error logging, Enterprise Replication logs this information as a failed operation.
- You cannot use the SPL routine or time stamp with SPL routine conflict-resolution rules.
- You cannot use update-anywhere replication; doing so can result in inconsistent conflict resolution.

Enterprise Replication logs bitmap information about the updated columns in the logical-log file. For more information, see the CDR record type in the logical-logs chapter in the *HCL® Informix® Administrator's Reference*.

Related reference

cdr define replicate on page 342

Related information

Controlling the replication of large objects on page 112

Setting Up Failed Transaction Logging on page 105

Conflict Resolution on page 40

Disk Space for Delete Tables on page 64

Using the IEEE Floating Point or Canonical Format

You can specify how the FLOAT and SMALLFLOAT data types are handled, depending on your platform.

About this task

You can specify sending this data in either IEEE floating point format or machine-independent decimal representation:

- Enable IEEE floating point format to send all floating point values in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating point format.

  To use IEEE floating point format, include the **--floatieee** option in your replicate definition.

  It is recommended that you define all new replicates with the **--floatieee** option.
- Enable canonical format to send floating-point values in a machine-independent decimal representation when you replicate data between dissimilar hardware platforms.

  To use canonical format, include the **--floatcanon** option in your replicate definition. The **--floatcanon** option is provided for backward compatibility only; it is recommended that you use the **--floatieee** option when defining new replicates.
- If you specify neither IEEE or canonical formats, Enterprise Replication sends FLOAT and SMALLFLOAT data types as a straight copy of machine representation. If you are replicating across different platforms, replicated floating-point numbers will be incorrect.

For more information, see Special Options on page 348.

Important: You cannot modify the replicate to change the **--floatieee** or **--floatcanon** options.

Related reference

cdr define replicate on page 342

Enabling Triggers

By default, when a replicate causes an insert, update, or delete on a target table, triggers associated with the table are not executed. However, you can specify that triggers are executed when the replicate data is applied by enabling triggers in the replicate definition.

About this task

To enable triggers, include the **--firetrigger** option in your replicate definition.

When you design your triggers, you can use the 'cdrsession' option of the DBINFO() function to determine if the transaction is a replicated transaction.

For information, refer to Triggers on page 26 and Special Options on page 348.

Related reference

cdr define replicate on page 342

Enabling code set conversion between replicates

You can enable code set conversion to allow replication of data between servers that use different code sets.

Before you begin

Prerequisites:

The table and column names must contain ASCII characters to convert a non-master replicate to a master replicate.

The servers must have UTF-8 code set transaction support enabled to replicate between server versions.

The target schema must allow for expansion due to code set conversion. For example, a CHAR(10) column in one code set might require 40 bytes in the converted code set. When replicating from non-UTF8 code set to UTF8 code set using master replicates (default from version 14.10 onwards), with target schema character type columns wider than the source schema character type columns, make sure such target is the master when defining the replicates, to avoid potential data truncation. Either use explicit -M <target_server> option or put target server first in the replicates' list of participants.

About this task

When code set conversion is enabled, character columns of the following data types are converted to UTF-8 (Unicode) when the row is copied into the transmission queue.

- CHAR
- VARCHAR
- NCHAR
- NVARCHAR
- LVARCHAR
- TEXT
- CLOB

When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set that is used on the target server. No attempt is made to convert character data contained within opaque data types, such as **TimeSeries** data types, user-defined data types, or DataBlade® module data types.No attempt is made to convert character data contained within opaque data types, such as user-defined data types or DataBlade® module data types.

To enable code set conversion between replicates, include the --UTF8=y option in your replicate definition.

To use the latest version of the Unicode library, set the GL_USEGLU environment variable in your server environment. The GL_USEGLU environment variable must be set to a value of 1 (one) in the database server environment before the server is started, and before the database is created.

If your table names or column names contain non-ASCII characters, you must manually create a shadow replicate and then swap the shadow replicate with the primary replicate using the cdr swap shadow command.

The autocreate option is not supported for replicates defined with --UTF8=y option when using the cdr realize template or cdr change replicate commands.

Code set conversion with the GLS library requires only those code set conversion files found in the `INFORMIXDIR/gls/cv9` directory.

- For US English, locales are handled automatically by the HCL Informix® Client Software Development Kit installation and setup.
- For other locales, you might need to explicitly provide the locale and conversion files.

Related reference

cdr swap shadow on page 475

Related information

GL_USEGLU environment variable on page

Global language support for replication on page 22

Configuring code set conversion between replicates

The examples in this topic show how to create replicate and template definitions while replicating data between databases that use different code sets.

When non-English characters are used for database, table, column, or owner names, each server must be added to the UTF-8 realize template definition by connecting to the server locally. Only one server at a time should be added to the replicate definition using the change replicate command. You cannot add multiple servers to a replication definition using the define repl command unless the database code set number is the same for all servers. The CLIENT_LOCALE environment variable must be set unless the database locale is en_us.819. Replicate and template names must be in English.

This example shows how to create and realize a template on two servers, named *node_1* and *node_2*. For this example, assume that *node_1* uses de_de.819 locale and *node_2* uses de_de.utf8 locale:

1. On node_1, run the following commands:

   ```
   export DB_LOCALE=de_de.819
   export CLIENT_LOCALE=de_de.819
   cdr define template set1 -C always -M g_node1 -S row -d testdb -a -A -R --UTF8=y
   cdr realize template set1 g_node1
   ```

2. On node_1 run the following command and wait for the Txns in queue count to go to zero.

   ```
   onstat -g rqm cntrlq
   ```

3. On node_2, run the following commands:

```
export DB_LOCALE=de_de.utf8
export CLIENT_LOCALE=de_de.utf8
cdr realize template set1 g_node2
```

The following steps show how to define a replicate when non-ASCII characters are used for table, column, owner, or database names. Before starting, ensure that the replicate name uses English ASCII characters and that the DB_LOCALE environment variable on the server is set to the same value as the locale of the participant being added.

1. Define the replicate with the first participant and then connect to the participant.
2. Add and connect to each additional participant, one participant at a time.
3. When all of the participants have been added, ensure that the control queue is empty and start the replicate definition.

   You can check the control queue message count using the onstat -g rqm cntrlq. Wait for the Txns in queue count to go zero.

The following example shows how to create a replicate definition between two servers to replicate data between de_de.819 and de_de.utf8 databases:

1. On server node_1, run the following commands:

   ```
   export DB_LOCALE=de_de.819
   export CLIENT_LOCALE=de_de.819
   cdr define repl german_repl  -M g_node1 -C always -S transaction
            -A -R -I --UTF8=y "testdb@g_node1:user1.table1" "select * from table1"
   ```

2. On node_1 run the following command and wait for the Txns in queue count to go to zero.

   ```
   onstat -g rqm cntrlq
   ```

3. On node_2, run the following commands:

   ```
   export DB_LOCALE=de_de.utf8
   export CLIENT_LOCALE=de_de.utf8
   cdr change repl -c node2 -a german_repl
              "testdb@g_node2:user1.table1" "select * from table1"
   ```

4. On node_2 run the following command and wait for the Txns in queue count to go to zero.

   ```
   onstat -g rqm cntrlq
   ```

5. Run the following command on either server:

   ```
   cdr start repl german_repl
   ```

Code set conversion errors

You can use the ATS and RIS files to identify problems that occur during code set conversion.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter on page 515

Each column in the RIS file begins with (W) if substitute characters were added to the column data or (E) if data was rejected because of a UTF-8 conversion failure.

Examples of conversion errors:

On the source server, a row of data fails conversion to UTF-8 code set.

Data sync error 63 is stored in an RIS file on the source server. The RIS file contains the row that failed to convert; the failed row is not converted and is not replicated on the target server. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:35:22
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:63 (Error while converting data from local database codeset to
    UTF8.) / SQL:0 / ISAM:0
LRH  Failed column list: charcol (W), ncharcol (E)
LRD 3|Lkqy|jvdHj@ifcjuWg|biLs|uk|RwvCZOpfpqruLAA|JloY|<27, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<4, TEXT, BB>|
    <18, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the source server, conversion from the local code set to UTF-8 resulted in the substitution of one or more characters in the row.

Data sync error 65 is stored in an RIS file on the source server, and the row is replicated. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:32:14
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
LRH Failed column list: charcol (W), ncharcol (W), vchar (W), nvchar (W),
    lvchar (W)
LRD 2|iU\VoJMZ|axhGRxKmDW|e@Xv|biLs|pyqasjUpAc{wCu|efM@}Vd|<22, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<36, TEXT, BB>
    |<15, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the target server, a row of data failed to convert from UTF-8 format to the local database code set.

Data sync error 64 is stored in an ATS/RIS file on the target server, and the row or transaction is aborted depending on the replicate scope. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 13:40:19
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 13:40:19
----------
```

```
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:64 (Error while converting data from UTF8 to local database
    codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: vchar (E)
RRD 3||jdicW|?|?|?|?|?|?
==========
TXH Transaction aborted
TXH ATS file:/usr4/nagaraju/utm/tmp/ats.utm_group_3.utm_group_1.D_3
    .110510_13:40:19.2 has also been created for this transaction
```

On the target server, conversion from UTF-8 to the local server code set resulted in the substitution of one or more characters in the row.

Data sync error 66 is stored in a warning RIS file on the target server, and the row is applied. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:3 / Name:utm_group_3 / CommitTime:11-05-10 13:13:58
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:13:58
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Insert
RRH CDR:66 (Substitute characters added while converting data
    from UTF8 to local database codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: charcol (W), ncharcol (W), vchar (W),
    nvchar (W), lvchar (W), textcol (W), textbcol (W), clobcol (W)
RRD 99||keI||m||<46, TEXT, PB 3 (utm_group_3) 1305051238
    (11/05/10 13:13:58)>|<68, TEXT, BB>|<13, CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Text and CLOB data conversion failures

If the conversion of text or CLOB data to UTF-8 fails on the source server then the blob buffer is marked with the appropriate error and the target servers create ATS/RIS files for these blob data conversion failures. Example text column conversion error:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 12:26:30
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 12:28:15
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
RRH  Failed column list: textcol (W)
LRD 2|<46, TEXT, PB 1 (utm_group_1) 1305048215 (11/05/10 12:23:35)
    >|<40, CLOB, SB 1305048215 (11/05/10 12:23:35)>
RRD 2|<44, TEXT, PB 1 (utm_group_1) 1305048390 (11/05/10
    12:26:30)>|<0(NoChange), CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Controlling the replication of large objects

You can control whether columns that contain unchanged large objects are always included in replicated rows.

By default, columns that contain unchanged large objects are not included in replicated rows. Large object columns are transmitted only when the data is changed.

You can specify to replicate columns that contain unchanged large objects by including the --alwaysRepLOBS=y option in the replicate definition. For example, if your replication system is designed for as a workflow, you must replicate all columns when you move data to the next site in the workflow.

If you want to change how large objects are replicated for an existing replicate, you must delete the replicate and then re-create the replicate.

Related reference

Replicate only changed columns on page 106

Related information

Workflow Replication on page 38

Replication to SPL routine

At target participant, ‘replication to SPL routine’ type replicate definition causes SPL routine to be executed instead of applying data to target table. Target participant for “replication to SPL routine? replicate definition can be configured to be same as source database, different database on the same server, or remote peer Enterprise Replication server. “Replication to SPL routine? replicate definition does not enforce the requirement to have primary key, unique index or ER key on the replicated table.

Note: Even though data is applied to stored procedure routine, target table definition must exist.

```
|--+--splname=spl_routine_name----+-------+---------------------+--------->
   |                              |       |               .-y-. |

'--'--jsonsplname=spl_routine_name-'      '- -- cascaderepl=-+-n-+-'
```

Long Form	Meaning
--splname	Stored procedure routine name to apply data to. SPL routine must exist at all participants. Column list for SPL routine extracted from replicate participant select statement column projection list.
--jsonsplname	Stored procedure routine name to apply data to. SPL routine and table definition must exist at all participants. Input argument for SPL routine must be a JSON document. --jsonsplname option is mutually exclusive to --splname option.
--cascaderepl	Enable cascade replication. Required if replication to SPL needs to be executed for the data applied through Enterprise Replication.

--splname option stored procedure argument list:

- Optype char(1) – operation type. Values include
  - I – Insert
  - U – Update
  - D – Delete
- Soucre_id integer – Source server id. Same as group id.
- Committime integer – Transaction commit time.
- Txnid bigint – Transaction id.
- userid - Userid of the user executing the IUD operation.
- session_id - Session id of the session executing the IUD operation.
- Before value column list.
- After value column list.

Note: Column list for SPL routine extracted from select statement projection list

--jsonsplname option SPL routine json argument

Attribute Name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner
database	Database name
txnid	8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.
operation_owner_id	User id of the user executing the IUD operation
operation_session_id	Session id of the session executing the IUD operation
commit_time	Transaction commit time for the event data.
rowdata	Row data in JSON document format. Data is returned in column name as key and column data as value.
before_rowdata	Before row data for “update? operation.

Example

Example: JSON Document format

```
{“operation?:"insert",?table?:"creditcardtxns",?owner?:"informix",?database?:"creditdb",?txnid?:2250573177224,
?
operation_owner_id?:201,?operation_session_id?:200,?commit_time?:1488243530,rowdata?:{“uid?:22,?cardid?:"6666-6
666-6666-6666",
?carddata?:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

{“opertion?:"update",table:"creditcardtxns",?owner?:"informix",?database?:"creditdb?,?txnid?:2250573308360,
?
operation_owner_id?:201,?operation_session_id?:202,?commit_time?:1488243832,?rowdata?:{uid:21,cardid:"7777-7777
-7777-7777",
?carddata?:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },
?before_rowdata?:{“uid?:21,?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams
 Club","Amount":200,"Date":2017-05-01T10:35:10.000Z  } }}

{“opertion?:"delete",?table?:"creditcardtxns",?owner?:"informix",?database?:"creditdb",?txnid?:2250573287760,
?
operation_owner_id?:201,?operation_session_id?:203,?commit_time?:1488243797,rowdata?:{“uid?:22,?cardid?:"6666-6
666-6666-6666",
?carddata?:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T13:35:06.000Z } }}
```

Asynchronous post commit trigger support

- Define loopback replication server. For more information, see Loopback Replication on page 251
- Create ‘replication to SPL’ type replicate with same “database and table? information for both source and target participants. Loopback server group name shall be specified with target participant definition

Example

```
cdr define replicate txn_repl -C always -S row -M group_1 -A -R --splname=logger4repl2spl
"stores@group_1:informix.transactions" "select * from transactions" "test@lb_group_1:informix.transactions"
 "select * from transactions?
```

Note: group_1 is the local server ER group, and lb_group_1 is the pseudo ER server group for loop back replication.

Prerequisites

- Enable Login for the tables.
- Ensure not to combine participant definitions that include table as the target, and SPL routine as the target.
- Define target table.
- Out-of-row data datatypes, TEXT, BYTE, BLOB, CLOB are not supported.

Example

Example 1: build staging table for data changes using --splname replicate attribute

```
create database storesdb with log;

create table transaction (uid int, bill_amount int);

create table changelog (optype int, srcdir int, committime int, txnid bigint, userid int, session_id int,
 uid_bef int,
 bill_amount_bef int, uid int, bill_amount int);

create procedure logger4repl2spl (opType char(1), srcid int, committime int, txnid bigint, userid int,
 session_id int,
uid_bef int, bill_amount_bef int, uid int, bill_amount_bef int)
```

```
insert into changelog values (opType, srcid, committime, txnid, userid, session_id, uid_bef, bill_amount_bef,
 uid, bill_amount);

end procedure;
```

Note: Require new code section for these shell commands:

```
$ cdr define replicate txn_repl -C always -S row -M group_1 -A -R --splname=logger4repl2spl
"storesdb@group_1:informix.transaction" "select * from transaction"
"storesdb@lb_group_1:informix.transaction" "select * from transaction?

$ cdr start replicate txn_repl
```

Example

Example 2: build staging table for data changes using --jsonsplname replicate attribute

```
create database storesdb with log;

create table transaction (uid int, bill_amount int);

create table inventory (inv_id int, inv_count int);

create table staging (data json);

create procedure logger4repl2spl (data json)
   insert into staging values (data);
end procedure;

$ cdr define replicate inv_repl -C always -S row -M group_1 -A -R --jsonsplname=logger4repl2spl
"storesdb@group1:informix.inventory" "select * from inventory" "storesdb@lb_group_1:informix.inventory" "select
 * from inventory?

$ cdr start replicate inv_repl

$ cdr define replicate txn_repl -C always -S row -M group_1 -A -R --jsonsplname=logger4repl2spl
"storesdb@group_1:informix.transaction" "select * from transaction"
"storesdb@lb_group_1:informix.transaction" "select * from transaction?

$ cdr start replicate txn_repl
```

Example

Example 3: Realtime aggregation framework

```
create database retaildb with log;

create table sales (customerid int, storeid int, bill_amount float);

create table sales_summary(storeid int , s_count int, s_sum float, s_avg float, s_min float, s_max float);

create procedure store_agg(opType char(1), srcid int, committime int, txnid bigint, userid int, session_id int,
customerid_bef int, storeid_bef int, bill_amount_bef float, customerid int, storeid_aft int, bill_amount float)
```

```
---- -----
----  ----
end procedure;

$ cdr define replicate sale_repl -C always -S row -M group_1 -A -R --serial --splname=store_agg
"retaildb@group_1:informix.sales" "select * from sales"
"retaildb@lb_group_1:informix.sales" "select * from sales"

$ cdr start replicate sale_repl
```

Note: For streaming aggregation, make sure to define replicate with –serial option to avoid executing multiple instances of the same SPL routine in parallel.

Example

Example 4: Calculate leader board on the fly based on changes to scores table

```
create table scores (playerid int, score int);
create table leaderboard(playerid int, score int);

create procedure leaderboard_spl(opType char(1), srcid int, committime int, txnid bigint, userid int,
 session_id int,
playerid_bef int, score_bef int, playerid_aft int, score_aft int)
… ….
end procedure;

$ cdr define replicate game_repl -C always -S row -M group_1 -A -R --serial --splname=leaderboard_spl
"gamedb@group_1:informix.scores" "select * from scores" "gamedb@lb_group_1:informix.scores" "select * from
 scores"

$ cdr start replicate game_repl
```

Define smart trigger on leaderboard table(mentioned in the example given above) and push out changes to application layer.

Note: For more information, see Smart Trigger.

Define replicate sets

When you define a replicate set, you specify the type of replicate set, the replicates that belong to the replicate set, and the frequency of replication for the member replicates.

To create a replicate set, use the cdr define replicateset command.

Enterprise Replication supports these types of replicate sets:

exclusive

Replicates can belong to only one replicate set. Include the --exclusive option in the cdr define replicateset command.

non-exclusive

Default. Replicates can belong to one or more non-exclusive replicate sets.

derived

A replicate set that is derived from an existing replicate set. For example, you can create a derived replicate set that contains replicates that must be remastered.

Related reference

cdr define replicateset on page 355

Exclusive Replicate Sets

If your replicated tables use referential integrity and are defined with time-based replication, you must create an *exclusive replicate set*. If your replicates use referential integrity and you plan to stop and start the replicate set, use an exclusive replicate set.

About this task

An exclusive replicate set has the following characteristics:

- All replicates in an exclusive replicate set have the same state and frequency settings. For more information, see cdr list replicateset on page 400.
- When you create the replicate set, Enterprise Replication sets the initial state of the replicate set to active.
- You can manage the replicates in an exclusive replicate set only as part of the set. Enterprise Replication does not support the following actions for the individual replicates in an exclusive replicate set:
    - Starting a Replicate on page 169
    - Stopping a Replicate on page 170
    - Suspending a Replicate on page 171
    - Resuming a Suspended Replicate on page 171
- Replicates that belong to an exclusive replicate set cannot belong to any other replicate sets.

To create an exclusive replicate set, use the --exclusive option with cdr define replicateset.

Important: You cannot change an exclusive replicate set to non-exclusive.

Related reference

cdr define replicateset on page 355

cdr define template on page 367

cdr resume replicate on page 440

Non-Exclusive Replicate Sets

About this task

By default, the **cdr define replicateset** command creates *non-exclusive* replicate sets.

A non-exclusive replicate set has the following characteristics:

- You can manage replicates that belong to a non-exclusive replicate set both individually and as part of the set.
- Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state.
- You should not use non-exclusive replicate sets for replicates that include tables that have referential constraints placed on columns.
- A replicate can belong to more than one non-exclusive replicate set.

Important: You cannot change a non-exclusive replicate set to exclusive.

Use non-exclusive replicate sets if you want to add a replicate to more than one replicate set. For example, you might want to create replicate sets to manage replicates on the target server, table, or entire database. To do this, create three non-exclusive replicate sets:

- A set that contains the replicates that replicate to the target server
- A set that contains the replicates on a particular table
- A set that contains all the replicates

In this scenario, each replicate belongs to three non-exclusive replicate sets.

Customizing the Replicate Set Definition

About this task

You can specify the replication frequency (Specifying Replication Frequency on page 104) for all the replicates when you define the replicate set. For example, to define the non-exclusive replicate set **sales_set** with the replicates **sales_fiji** and **sales_tahiti** and specify that the members of **sales_set** replicate at 4:00 a.m. every day, enter:

```
cdr define replicateset --at 4:00 sales_set sales_fiji \
   sales_tahiti
```

To define the exclusive replicate set **dev_set** with the replicates **dev_pdx** and **dev_lenexa** and specify that the members of **dev_set** replicate at 5:00 p.m. every day, enter:

```
cdr define replicateset -X --at 17:00 dev_set dev_pdx\
   dev_lenexa
```

Important: For replicates that belong to an exclusive replicate set, you cannot specify the frequency individually for replicates in the set.

For more information, see cdr define replicateset on page 355.

Initially Synchronizing Data Among Database Servers

About this task

Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate.

You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

The **cdr start replicate** and **cdr start replicateset** commands provide options to perform an initial synchronization for the replicates you are starting. All of the rows that match the replication criteria will be transferred from the source server to the target servers. If you are starting a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

Use the **--syncdatasource** (**-S**) option of the **cdr start replicate** or **cdr start replicateset** command to specify the source server for synchronization. Any existing rows in the specified replicates are deleted from the remote tables and replaced by the data from the node you specify using **-S**.

The **--extratargetrows** option of the **cdr start replicate** or **cdr start replicateset** commands specifies how to handle rows found on the target servers that are not present on the source server. You can specify to remove rows from the target, keep extra rows on the target, or replicate extra rows from the target to other participants.

If you use the **cdr start replicate** or **cdr start replicateset** command to specify a subset of servers on which to start the replicate (or replicate set), that replicate (or replicate set) must already be active on the source server. The source server is the server you specify with the **-S** option. For example, for the following command, **repl1** must already be active on **serv1**:

```
cdr start repl repl1 ... -S serv1 serv2 serv3
```

When you start a replicate (or replicate set) for participants on all servers, the replicate does not need to be active on the source server. So, for the following command, **repl1** does not need to be active:

```
cdr start repl1 ... -S serv1
```

When Enterprise Replication performs initial data synchronization, it keeps track of discrepancies between the constraints set up on source and target server tables. Rows that fail to be repaired due to these discrepancies are recorded in the ATS and RIS files.

If replication fails for some reason and data becomes inconsistent, there are different ways to correct data mismatches between replicated tables while replication is active. The recommended method is direct synchronization. You can also repair data based on an ATS or RIS file. Both of these methods are described in Resynchronizing Data among Replication Servers on page 177.

Related information

Repair and Initial Data Synchronization on page 8

Load and unload data on page 82

Set up replication through templates

Enterprise Replication provides templates to allow easy setup and deployment of replication for clients with large numbers of tables to replicate. A template uses schema information about a database, a group of tables, columns, and replication keys to define a group of master replicates and a replicate set.

Do not use a template if you want to use time-based replication.

You create a template by running the cdr define template command and then you instantiate the template on the servers where you want to replicate data by running the cdr realize template command.

Templates set up replication for all the columns in the table. Templates are useful for setting up large-scale replication environments. If you want a participant to contain a partial row (just some columns in the table), you can either set up replication manually, or, after you realize a template you can run the cdr remaster command to restrict the query.

Defining Templates

About this task

You define a template using the **cdr define template** command, with which you can specify which tables to use, the database and server they are located in, and whether to create an exclusive or non-exclusive replicate set. Table names can be listed on the command line or accessed from a file using the **--file** option, or all tables in a database can be selected.

Important: A template cannot define tables from more than one database.

Specify that the replicate set is exclusive if you have referential constraints on the replicated columns. Also, if you create an exclusive replicate set using a template, you do not need to stop the replicate set to add replicates. For more information about exclusive replicate sets, see Define replicate sets on page 117.

A template defines a group of master replicates and a replicate set.

You can use the **cdr list template** command from a non-leaf node to view details about the template, including the internally generated names of the master replicates. These are unique names based on the template, the server, and table names.

Realizing Templates

About this task

After you define a template using the **cdr define template** command, use the **cdr realize template** command to instantiate the template on your Enterprise Replication database servers. The **cdr realize template** command first verifies that the tables on each node match the master definition used to create the template. Then, on each node, it adds the tables defined in the template as participants to master replicates created by the template.

If a table on a server has additional columns to those defined in the template, those columns are not considered part of the replicate.

If a table does not already exist on a server where you realize the template, you can choose to create it, and it is also added to the replicate.

Also, at realization time, you can also choose to synchronize data among all servers.

Verifying Participants without Applying the Template

About this task

The **--verify** option allows you to check that a template's schema information is correct on all servers before actually instantiating the template.

Synchronizing Data Among Database Servers

About this task

Use the **--syncdatasource** option to specify a server to act as the source for data synchronization on all servers where you are realizing the template. The server listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.

Improve Performance During Synchronization

You can speed up a synchronization operation by temporarily increasing the size of the send queue.

Enterprise Replication uses the value of the CDR_QUEUEMEM configuration parameter as the size of the send queue during a synchronization operation. To increase the size of the send queue during a particular synchronization operation, use the **--memadjust** option.

In addition to controlling memory during initial synchronization, you can also control memory consumption when you realize a template and perform a direct synchronization.

Create tables automatically

You automatically create tables in the template definition if they do not exist on a server.

Include the --autocreate option in the cdr realize template command to automatically create tables. You cannot use the --autocreate option for tables that contain user-defined data types.

Use the --dbspace option to specify a dbspace for table creation.

Note: Tables that are created by --autocreate option do not automatically include non-replicate key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. You must manually create these objects.

Other synchronization options

Several other options to the cdr realize template command can affect how synchronization occurs.

About this task

You can use the **--applyasowner** option to realize a table by its owner rather than the user **informix**.

The **--extratargetrows** option specifies whether to delete, keep, or merge rows found on target servers that are not present on the source server during the synchronization operation.

The **--target** option defines whether target servers are receive-only (when target servers are defined as receive-only, updates made on those servers are not propagated).

The **--mode** option defines whether servers only receive or only send data.

Changing Templates

About this task

You cannot update a template. To adjust a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

Template Example

This example illustrates a scenario in which one template is created and realized on two servers, then realized on a third server.

About this task

The template **Replicateset1** is defined on three tables in the **college** database: **staff**, **students**, and **schedule**. The template is first realized on the servers **g_cdr_ol_1** and **g_cdr_ol_2**.

This procedure is performed as follows:

1. Define the template **Replicateset1** on the **staff**, **students**, and **schedule** tables of the **college** database:

   ```
   cdr define template -c g_cdr_ol_1 Replicateset1 -M g_cdr_ol_1\
    -C "timestamp" -A -R -d college testadm.staff testadm.students\
    testadm.schedule
   ```

   This command also creates the replicate set **Replicateset1**.

2. Realize the template on the server **g_cdr_ol_1**:

   ```
   cdr realize template -c g_cdr_ol_1 Replicateset1 "college@g_cdr_ol_1"
   ```

3. Realize the template on server **g_cdr_ol_2** and synchronize the data with server **g_cdr_ol_1**:

```
cdr realize template -c g_cdr_ol_2 -u -S g_cdr_ol_1 \
Replicateset1 "college@g_cdr_ol_2"
```

4. Realize the **Replicateset1** template on a new server **g_cdr_ol_3** and synchronize the data with server **g_cdr_ol_1**. The **g_cdr_ol_3** server participant is automatically added to all replicates within the **Replicateset1** template:

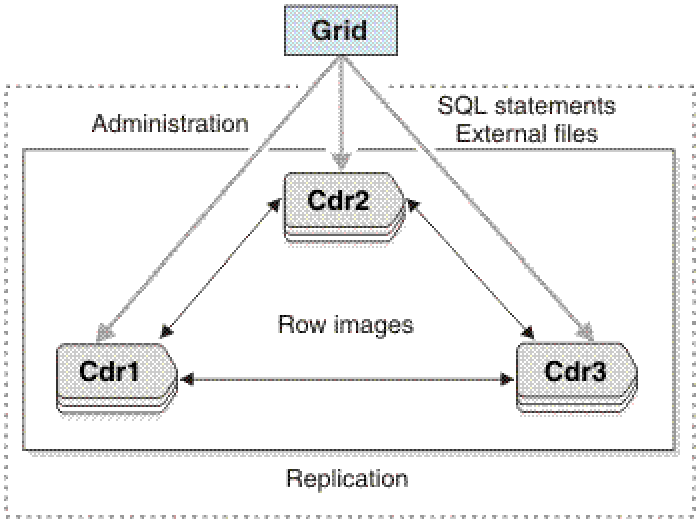
```
cdr realize template -c g_cdr_ol_1 -u -S g_cdr_ol_1\
  Replicateset1 "g_cdr_ol_3"
```

Grid setup and management

A grid is a set of replication servers that are configured to simplify administration. When you run SQL data definition statements from within a grid context on a grid server, the statements propagate to all servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can choose to set up replication automatically when you create a table through a grid. You can propagate external files to other servers in the grid.

SQL statements are not replicated by Enterprise Replication. Enterprise Replication replicates the row images that are the results from SQL statements. The grid propagates SQL statements, but does not, by default, propagate the results of propagated SQL statements. The following illustration shows three replication servers, named **Cdr1**, **Cdr2**, and **Cdr3**, that replicate row images between each other, while the grid propagates SQL statements and administration commands.

Figure 17. Replication of rows as a grid propagates SQL statements to each server.



A grid can be useful if you have multiple replication servers and you often perform the same tasks on every replication server. The following types of tasks can be run through the grid:

- Creating replicated tables. When you create a replicated table through a grid, the other tasks for setting up replication are completed automatically: a replicate is created for the table, participants are defined for each replication server, and the replicate is added to the grid replicate set.
- Administering servers, for example, adding chunks, removing logical logs, or changing configuration parameter settings
- Updating the database schema, for example, altering, adding, or removing tables
- Running or creating stored procedures or user-defined routines

- Updating data, for example, purging old data or updating values that are based on conditions
- Altering a replicate definition when you alter a replicated table
- Copying external files to grid servers

For example, suppose that you have 100 replication servers and must create a table. You must fragment the table into two new dbspaces. You also must create a new stored procedure to run on the table. With a grid, you would run four commands to perform these tasks on all 100 replication servers, instead of running 400 commands. The command to create the table can also specify that the data in that table is replicated.

You can control the security of the grid by authorizing which users can run grid routines on which servers. You can monitor the results of grid routines and rerun any failed routines on the appropriate servers.

You can configure Connection Managers to route client connection requests to the replication servers of a grid, based on one of the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

Related information

Connection management through the Connection Manager on page

Example of setting up a replication system with a grid

This comprehensive example sets up a replication domain, creating a grid, creating a database, creating a replicated table, and loading data.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix® database server installed, but no databases defined.

1. On all servers, set the CDR_QDATA_SBSPACE configuration parameter.
2. Edit the `sqlhosts` files on all four servers so that they each have the following information:

```
#dbservername   nettype   hostname              servicename   options
gserv1          group     -                     -             i=143
serv1           ontlitcp  ny.usa.com            1230          g=gserv1
gserv2          group     -                     -             i=144
serv2           ontlitcp  tokyo.japan.com       1231          g=gserv2
gserv3          group     -                     -             i=145
serv3           ontlitcp  rome.italy.com        1232          g=gserv3
gserv4          group     -                     -             i=146
serv4           ontlitcp  perth.australia.com   1233          g=gserv4
```

3. Define each server as a replication server by running the cdr define server command:

```
cdr define server -c gserv1 -I gserv1
cdr define server -c gserv2 -S gserv1 -I gserv2
cdr define server -c gserv3 -S gserv1 -I gserv3
cdr define server -c gserv4 -S gserv1 -I gserv4
```

4. Create a grid that includes all replication servers in the domain as members of the grid:

   ```
   cdr define grid grid1 --all
   ```

5. Authorize the user **bill** to run commands on the grid and designate the server **gserv1** as the source server from which grid commands can be run:

   ```
   cdr enable grid --grid=grid1 --user=bill --node=gserv1
   ```

   Tip: User **informix** does not have permission to run grid operations unless you include it in the user list.

6. Run cdr list grid to see the grid configuration:

   ```
   Grid                Node                User
   ------------------  ------------------  ----------------
   grid1               gserv1*             bill
                       gserv2
                       gserv3
                       gserv4
   ```

   The asterisk indicates that **gserv1** is the source server for the grid.

7. Run the cdr list replicateset command to see the grid replicate set information:

   ```
   Ex  T  REPLSET                 PARTICIPANTS
   -----------------------------------------------
   Y   Y   grid1
   ```

   The replicate set has the same name as the grid. It does not yet contain any participants.

8. Create two dbspaces named **dbsp2** and **dbsp3** in which to fragment a table:

   ```
   database sysmaster;

   EXECUTE FUNCTION ifx_grid_function('grid1',
     'task("create dbspace","dbsp2",
             "/db/chunks/dbsp2","2G","0")');

   EXECUTE FUNCTION ifx_grid_function('grid1',
     'task("create dbspace","dbsp3",
             "/db/chunks/dbsp3","8G","0")');
   ```

   The dbspaces are created on all four servers.

9. Create database named **retail** and a table named **special_offers** with replication enabled:

   ```
   database sysmaster;

   EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

   CREATE DATABASE retail WITH LOG;

   CREATE TABLE special_offers(
       offer_description varchar(255),
       offer_startdate   date,
   ```

```
    offer_enddate     date,
    offer_rules       lvarchar,
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2,
    REMAINDER IN dbsp3;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

10. Run the cdr list grid --verbose grid1 command to see information about the statements on each server:

```
Grid               Node               User
------------------ ------------------ ----------------
grid1              gserv1*            bill
                   gserv2
                   gserv3
                   gserv4
Details for grid grid1

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE DATABASE retail WITH LOG;
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2
    REMAINDER IN dbsp3;
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59
```

Both statements succeeded on all four servers.

11. Run cdr list replicate to see the replicate information:

```
CURRENTLY DEFINED REPLICATES
---------------------------------------------
REPLICATE:        gserv1_1
STATE:            Active
CONFLICT:         Timestamp
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      retail:bill.special_offers
OPTIONS:
REPLTYPE:         Master,Grid
```

The replicate was created and is active.

12. Run the cdr list replicate brief gserv1_1 command to see the participants:

```
REPLICATE  TABLE                              SELECT
----------------------------------------------------------------
gserv1_1   retail@gserv1:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
```

13. Load data onto one of the replication servers and Enterprise Replication replicates the data to the other servers. For more information, see Load and unload data on page 82.

Related reference

cdr enable grid on page 386

cdr list grid on page 391

cdr list replicateset on page 400

Related information

Adding a replication server to a grid by cloning on page 132

Connection management for client connections to participants in a grid on page 145

sqlhosts connectivity information on page

Example of rolling out schema changes in a grid

You can roll out schema changes to replicated tables through a grid without shutting down your applications.

Suppose that you have a grid replicate set named **gridset** that contains 12 replicates, each of which represents a different table. You want to alter the data types of columns in five tables. The grid contains four servers.

To roll out schema changes without application downtime:

1. Change any connections from the original application to the replication server named **cdr1** to connect to the replication server named **cdr2**.
2. On the **cdr1** server, connect to the **stores_demo** database, connect to the grid, and alter the five tables:

```
dbaccess stores_demo -
EXECUTE PROCEDURE ifx_grid_connect('grid1', 'gridset', 4);
SET LOCK MODE TO WAIT 120;
ALTER TABLE customer ADD prefix (char15);
ALTER TABLE items MODIFY order_num (bigint);
ALTER TABLE stock MODIFY description (lvarchar);
ALTER TABLE cust_calls ADD call_descr2 (lvarchar);
ALTER TABLE manufact MODIFY manu_name (char32);
```

The ifx_grid_connect() procedure changes the tables on **cdr1** but delays the propagation of the changes to the other replication servers.

3. Update the application to reflect the new schema for the five tables and connect to the server **cdr1**.
4. Close the connections from the original application.
5. On the server **cdr1**, propagate schema changes to the other replication servers by running the following statement:

   ```
   EXECUTE FUNCTION ifx_grid_release('grid1', 'gridset');
   ```

6. On the server **cdr1**, create a derived replicate set named **alterSet** that contains the altered tables by running the following command:

   ```
   cdr define replicateset --needRemaster=gridset alterSet
   ```

7. From the server **cdr1**, remaster the altered tables on all replication servers by running the following command:

   ```
   cdr remaster replicateset --master=cdr1 alterSet
   ```

8. From the server **cdr1**, synchronize the data on all replication servers by running the following command:

   ```
   cdr check replicateset --replset=alterSet --repair --master=cdr1 --all
   ```

9. On the server **cdr1**, drop the derived replicate set by running the following command:

   ```
   cdr delete replicateset alterSet
   ```

Related information

Altering multiple tables in a replicate set on page 190

Propagating database object changes on page 141

Creating a grid

You can create a grid based on an existing replication domain. You must authorize users who can run grid routines, and designate a server from which to run grid routines.

About this task

You must be connected to a replication server in the domain that contains the servers that you want to include in the grid.

To create a grid:

1. Specify a name for the grid and the servers to include in the grid by running the cdr define grid command. For example, the following command creates a grid named **grid1** and adds all replication servers in the domain as members of the grid:

   Example

   ```
   cdr define grid grid1 --all
   ```

2. Authorize users to run commands on the grid and designate a server from which grid commands can be run by running the cdr enable grid command. For example, the following command authorizes the user **bill** to run commands on the server **gserv1**:

   Example

```
cdr enable grid --grid=grid1 --user=bill --node=gserv1
```

Result

Only authorized users can run grid routines on authorized servers. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Related reference

cdr define grid on page 338

cdr enable grid on page 386

Grid maintenance

You can adjust grid membership, change user or server authorization to run grid routines, and delete grid-routine history from the **syscdr** database.

To see information about the grid, such as, which servers can run grid routines and the status of routines that are run on the grid servers, run the cdr list grid command.

If you remove a server from your replication domain, remove the server from your grid. The following example removes a replication server named **gserv1** from the grid **grid_1**:

```
cdr change grid grid_1 --delete gserv1
```

You cannot drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.

You cannot rename a replicated database. You must manually rename the database on each participant server by using the cdr remaster command.

To change which users can run routines on the grid or which servers are authorized to run grid routines, run the cdr enable grid and cdr disable grid commands. For example, to change the authorized server from **gserv1** to **gserv2** and authorize the user **srini**, run the following commands:

```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2 --user=srini
```

To delete the history of grid routines, run the ifx_grid_purge() procedure. You must occasionally purge information about completed grid routines to prevent the **syscdr** database from growing too large.

Related reference

cdr change grid on page 296

cdr disable grid on page 382

cdr enable grid on page 386

cdr list grid on page 391

ifx_grid_purge() procedure on page 540

Viewing grid information

You can view information about a grid and whether a replicate or replicate set belongs to a grid.

To view information about a grid:

Run the cdr list grid command. For example, the following command shows the servers and authorized users for a grid named **grid1**:

Example

```
cdr list grid grid1
```

The output for this command might be:

```
Grid               Node               User
------------------ ------------------ ----------------
grid1              gserv1*            bill
                   gserv2
                   gserv3
                   gserv4
```

The user **bill** is authorized to run grid commands on the server **gserv1**.

What to do next

You can see whether a replicate is a member of a grid replicate set by running the cdr list replicate command or the onstat -g cat repls command. You can also query the **syscdrrepl** SMI table. The following example output of the cdr list replicate command shows that the replicate is a master replicate and a member of a grid replicate set:

```
CURRENTLY DEFINED REPLICATES
---------------------------------------------
REPLICATE:        grid_6553604_100_3
STATE:            Active ON:g_delhi
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      tdb:nagaraju.t1
OPTIONS:          row,ris,fullrow
REPLID:           6553605 / 0x640005
REPLMODE:         PRIMARY  ON:gserv1
APPLY-AS:         INFORMIX ON:gserv1
REPLTYPE:         Master,Grid
```

Related reference

cdr list replicateset on page 400

cdr list replicate on page 395

onstat -g cat: Print ER global catalog information on page 548

The syscdrrepl Table on page 588

Adding replication servers to a grid

There are multiple ways to add a replication server to a grid.

You can add a replication server to a grid in the following ways:

- Run the cdr change grid command.
- Clone an existing replication server in the grid.

Adding a replication server to a grid by running cdr change grid

You can add a replication server to a grid by running the cdr change grid command.

To add a replication server to a grid:

Run the cdr change grid command.

Example

For example, to add a replication server named **gserv3** to the grid **grid1**, run the following command:

```
cdr change grid grid1 --add=gserv3
```

Results

To see information about the grid, such as, which servers can run grid routines and the status of routines that are run on the grid servers, run the cdr list grid command.

Related reference

cdr change grid on page 296

Adding a replication server to a grid by cloning

You can add a new server to a grid by cloning an existing replication server in the grid.

Before you begin

The server you are adding to the grid must have the same hardware and operating system as the source server that you are cloning.

To add a server to a grid:

Clone an existing replication server in the grid by using the ifxclone utility with the --disposition=ER option. This process is described in Adding a server to the domain by cloning a server on page 97.

Example

The following example adds a fifth server, named **serv5**, to an existing replication domain and to a grid named **grid1**. The server **serv1** is used as the source server.

1. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_COPY configuration parameter to `1` in the `onconfig` file.
2. On the **serv5** servers, complete the ifxclone prerequisites for all servers, such as setting the required configuration parameters and environment variables.

   Set these environment variables:

   - **INFORMIXDIR**
   - **INFORMIXSERVER**
   - **INFORMIXSQLHOSTS**
   - **ONCONFIG**

   Set these configuration parameters to the same values on the **serv5** server as on the **serv1** server:

   - DRAUTO
   - DRINTERVAL
   - DRTIMEOUT
   - LOGBUFF
   - LOGFILES
   - LOGSIZE
   - LTAPEBLK
   - LTAPESIZE
   - ROOTNAME
   - ROOTSIZE
   - PHYSBUFF
   - PHYSFILE
   - STACKSIZE
   - TAPEBLK
   - TAPESIZE
3. On the **serv5** server, create all of the chunks that exist on the **serv1** server.
4. On the **serv5** server, run the ifxclone command with the --disposition=ER option to clone the data and the configuration of the **serv1** server onto the **serv5** server and the --createchunkfile command to create the necessary chunks:

   ```
   ifxclone --trusted --source=serv1 --sourceIP=111.222.333.444
   --sourcePort=1230 --target=serv5 --targetIP=111.222.333.777
   --targetPort=1234 --disposition=ER --createchunkfile
   ```

   ```
   ifxclone --trusted --source=serv1 --sourceIP=111.222.333.444
   --sourcePort=1230 --target=serv5 --targetIP=111.222.333.777
   --targetPort=1234 --disposition=ER
   ```
5. Edit the `sqlhosts` files on all five servers in the domain so that they each have the following information:

   ```
   #dbservername   nettype   hostname             servicename   options
   gserv1          group     -                    -             i=143
   ```

```
serv1           ontlitcp  ny.usa.com            1230          g=gserv1
gserv2          group     -                     -             i=144
serv2           ontlitcp  tokyo.japan.com       1231          g=gserv2
gserv3          group     -                     -             i=145
serv3           ontlitcp  rome.italy.com        1232          g=gserv3
gserv4          group     -                     -             i=146
serv4           ontlitcp  perth.australia.com   1233          g=gserv4
gserv5          group     -                     -             i=147
serv5           ontlitcp  helsinki.finland.com  1234          g=gserv5
```

The server **serv5** is automatically added to the grid **grid1**.

Related reference

cdr change grid on page 296

Related information

Example of setting up a replication system with a grid on page 125

Adding a server to the domain by cloning a server on page 97

onconfig Portal: Configuration parameters by functional category on page

ENABLE_SNAPSHOT_COPY configuration parameter on page

The ifxclone utility on page

Adding an externally created replicate into a grid replicate set

If a replicate is created external to a grid, it can still be added to a grid replicate set.

You can add an existing replicate to a grid replicate set in the following ways:

- Run the cdr change replicateset command.
- Alter a replicate through a grid.

Adding an existing replicate to a grid replicate set by using cdr change replicateset

You can use the cdr change replicateset command to add replicates created outside of a grid environment to a grid replicate set.

Before you begin

Before you begin, you must verify the following items:

- All the replicate participants are members of the grid. Replicate participants must include every member node of the grid, and no additional participants.
- Each replicate participant's information refers to the same database, owner, table name and SELECT statement.
- The replicated table schema is the same among all participants.
- The replicate does not belong to an exclusive replicate set.

To add a replicate to a grid replicate set by using the cdr remaster command:

1. Use the cdr remaster command to convert the replicate to a mastered replicate.
2. Run the cdr change replicateset command with the --add option and specifying the grid replicate set.

   Example

   For example, the following command adds a replicate named **vendors** to the **grid1** grid replicate set:

   ```
   cdr change replicateset --add grid1 vendors
   ```

Results

When you run the cdr list replicate command, the `REPLTYPE` field shows `Grid`.

Related reference

cdr change replicateset on page 302

cdr list replicate on page 395

Related information

sqlhosts connectivity information on page

Adding an existing replicate to a grid replicate set by altering a table on page 135

Adding an existing replicate to a grid replicate set by altering a table

You can alter replicated tables through a grid even if the replicate was not created through a grid. Altering a replicated table through a grid adds the replicate to the grid replicate set.

Before you begin

Before you begin, you must verify the following items:

- All the replicate participants are members of the grid. Replicate participants must include every member node of the grid, and no additional participants.
- Each replicate participant's information refers to the same database, owner, table name and SELECT statement.
- The replicated table schema is the same among all participants.
- The replicate does not belong to an exclusive replicate set.

About this task

To alter a replicated table through a grid:

1. Connect to the grid by running the ifx_grid_connect() procedure with the *ER_enable* argument set to `1`.
2. Run an ALTER TABLE statement.
3. Disconnect from the grid by running the ifx_grid_disconnect() procedure.

Results

The replicate is automatically remastered.

Example

The following example adds a new column to the **special_offers** table and remasters the replicate on all participants that are members of the grid:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
   offer_exceptions varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related reference

ifx_grid_connect() procedure on page 527

Related information

Removing replicated columns on page 191

Adding an existing replicate to a grid replicate set by using cdr change replicateset on page 134

Creating replicated tables through a grid

You can automatically create a replicate and start replication when you create a table through the grid.

Before you begin

If the table you are creating is a typed table, you must define a primary key.

If you plan to create a table with a **TimeSeries** column, all grid servers must be running Informix® version 12.10 or later.

About this task

When you enable replication while creating a table through a grid, replication is set up in the following way:

- A replicate is created for the table. The replicate name is based on the name of the source server. Use the cdr list replicate command to see the name.
- All servers that are members of the grid are included as participants in the replicate.
- The replicate is included in a replicate set that has the same name as the grid.
- The conflict resolution rule for the replicate is time stamp if you include the WITH CRCOLS clause. Otherwise, the conflict resolution rule is always apply.
- The ERKEY shadow columns are automatically added to the table.
- All other replicate properties are the same as the default properties of a replicate created through a template.

To set up replication:

1. Connect to the grid by running the ifx_grid_connect() procedure with the *ER_enable* argument set to `1`.
2. Run a CREATE TABLE statement. Include the WITH CRCOLS clause if you want time stamp conflict resolution.
3. Disconnect from the grid by running the ifx_grid_disconnect() procedure.

Example

The following example creates a table with replication enabled that uses the time stamp conflict resolution rule:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

What to do next

If you need to alter or delete a database object that you created through a grid, perform those operation from within a grid context. For example, do not create a table from within a grid and then delete the table on one of the replication servers outside of a grid context. Instead, delete the table through the grid.

Related reference

ifx_grid_connect() procedure on page 527

cdr define template on page 367

Related information

Preparing tables without primary keys on page 79

Conflict resolution rule on page 40

Enabling replication within a grid transaction

You can enable replication within a transaction that is run in the context of the grid.

About this task

By default, the results of transactions run in the context of the grid are not also replicated by Enterprise Replication. In certain situations you might want to both propagate a transaction to the servers in the grid and replicate the results of the transaction.

To enable replication within a transaction:

1. Connect to the grid with the ifx_grid_connect() procedure.
2. Create a procedure that performs the following tasks:
    a. Defines a data variable for the Enterprise Replication state information.
    b. Runs the ifx_get_erstate() function and save its result in the data variable.
    c. Enables replication by running the ifx_set_erstate() procedure with an argument of `1`.
    d. Runs the statements that you want to replicate.
    e. Resets the replication state to the previous value by running the ifx_set_erstate() procedure with the name of the data variable.

3. Disconnect from the grid with the ifx_grid_disconnect() procedure.
4. Run the newly-defined procedure by using the ifx_grid_procedure() procedure.

Example

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named low_inventory() that creates a low inventory report exists on all the servers in the grid named **grid1**. The following example creates a new procedure named xqt_low_inventory() that enables replication for the low_inventory() procedure, and then runs the low_inventory() procedure:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
CREATE PROCEDURE xqt_low_inventory()
 DEFINE curstate integer;
 EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
 EXECUTE PROCEDURE ifx_set_erstate(1);
 EXECUTE PROCEDURE low_inventory();
 EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
EXECUTE PROCEDURE ifx_grid_disconnect();
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'xqt_low_inventory()');
```

The following events occur in this example:

1. The ifx_grid_connect() procedure connects to the **grid1** grid so that the xqt_low_inventory() procedure is propagated to all the servers in the **grid1** grid.
2. The xqt_low_inventory() procedure defines a data variable called **curstate** to hold the Enterprise Replication state information.
3. The ifx_get_erstate() function obtains the Enterprise Replication state and stores it in the **curstate** variable. The ifx_set_state() procedure enables replication.
4. The low_inventory() procedure is run.
5. The replication state is reset back to its original value.
6. The connection to the grid is closed by the ifx_grid_disconnect() procedure.
7. The ifx_grid_procedure() procedure runs the xqt_low_inventory() procedure on all the servers in the grid and the result of the low_inventory() procedure is replicated like any normal updating activity.

Related reference

ifx_set_erstate() procedure on page 546

ifx_get_erstate() function on page 526

Propagating updates to data

You can change your data through a grid routine and propagate the changes to all the servers in the grid.

About this task

You can propagate updates to data on servers in the grid. By default, changes to data that are propagated through the grid are treated the same as changes to data that are made by Enterprise Replication apply threads: they are not replicated again. For example, if you propagate a DELETE statement through the grid to remove old data, you would not want the resulting deleted rows to be replicated as well. Although you can use the grid to run a DML statement, in general, use Enterprise Replication to replicate changes to replicated data.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate an SQL statement or a stored procedure that updates data, run the ifx_grid_execute() procedure with the DML statements or the stored procedure as the second argument.

Example

Examples

Example 1: Reduce the price of products with low sales

In the following example, the ifx_grid_execute() procedure runs SQL statements that reduce the price of wool overcoats in stores that did not sell an overcoat in the last week:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
  'UPDATE price_table SET price = price * 0.75
   WHERE item =
   (SELECT item FROM inventory i, sales s
     WHERE i.description = "Wool Overcoat"
        AND  i.item = s.item
        AND  s.recent_sale_date <
    extend (current - Interval(7) DAY))');
```

Example 2: Purge old data

The following example purges all sales records before 2010:

```
Database retail_db;
 EXECUTE PROCEDURE ifx_grid_execute('grid1',
    'DELETE FROM sales WHERE sales_year < 2010');
```

Example 3: Run a low inventory report

The following example runs an existing stored procedure named low_inventory():

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'low_inventory()');
```

Related reference

ifx_grid_execute() procedure on page 533

Administering servers in the grid with the SQL administration API

You can run SQL administration API commands in grid routines to perform administrative tasks on all servers in the grid.

About this task

The grid must exist and you must run the grid routines as an authorized user from an authorized server and while connected to the **sysadmin** database.

To propagate an SQL administration API command:

1. Run the ifx_grid_function() function with the SQL administration API command as the second argument.
2. Check the return code of the SQL administration API command to determine if it succeeded by running the cdr list grid command.
   The cdr list grid command shows the return code. The status of the ifx_grid_function() function can be ACK, which indicates success, even if the SQL administration API command failed.

Results

Example

Examples

The following examples must be run in the **sysadmin** database.

Example 1: Change a configuration parameter setting

The following example sets the maximum size of the log staging directory to 100 KB on all the servers in the grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
  'admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")');
```

The output of the cdr list grid command shows that the admin() function succeeded because the return codes are positive numbers:

```
Grid                Node                User
------------------- ------------------- ----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")
ACK cdr1 2010-05-27 15:21:57
    '110'
ACK cdr2 2010-05-27 15:21:58
    '111'
ACK cdr3 2010-05-27 15:21:58
    '112'
```

Example 2: Create a new dbspace

The following example creates a new dbspace on all the servers in the **grid1** grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
  'task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0")');
```

The output of the cdr list grid command shows that the task() function failed:

```
Grid               Node               User
------------------ ------------------ ----------------
grid1              cdr1*              bill
                   cdr2
                   cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0"
ACK cdr1 2010-05-27 15:21:57
    'Unable to create file /db/chunks/dbsp2'
ACK cdr2 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
ACK cdr3 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
```

Related reference

ifx_grid_function() function on page 534

ifx_grid_execute() procedure on page 533

Propagating database object changes

You can create or alter database objects by running DDL statements while connected to the grid and propagate the changes to all the servers in the grid.

About this task

You can propagate creating, altering, and dropping database objects to servers in the grid. For example, you can create a database or table or alter an existing database or table. You can also create stored procedures and user-defined routines.

You can choose to run the DDL statements on the local server and defer the propagation of the DDL statements to the other grid servers. Deferred propagation of DDL statements can be useful when you are rolling out schema changes or performing a rolling upgrade.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate DDL statements:

1. Connect to the grid by running the ifx_grid_connect() procedure.
2. Run one or more SQL DDL statements.
3. Disconnect from the grid by running the ifx_grid_disconnect() procedure.

Results

If you deferred the propagation of DDL statements, you can propagate them by running the ifx_grid_release() function, or remove them by running the ifx_grid_remove() function.

Example

Example

Suppose that you have a retail shop with a website. You replicate your data to several other locations for web applications. You want to be able to quickly and easily create, drop, and update tables. You create a grid named **grid1**, from which you can update the database schema for all servers in one step. The following example creates a table for special offers in the **prod_db** database:

```
Database  prod_db;

 EXECUTE PROCEDURE ifx_grid_connect('grid1');

  CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules        lvarchar);
 EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related reference

Example of rolling out schema changes in a grid on page 128

ifx_grid_connect() procedure on page 527

ifx_grid_disconnect() procedure on page 532

Propagating external files through a grid

You can copy non-database, external files to the servers within a grid.

About this task

The ifx_grid_copy() procedure copies files from a directory on the source server to a specified destination on all servers in a grid. You specify the source directory on the source server by setting the GRIDCOPY_DIR configuration parameter to the location of the file to copy. You also set the GRIDCOPY_DIR configuration parameters on each of the destination servers to specify the directory to which the file is copied. The source directory can be different than the destination directory.

The file is copied to all of the servers within the grid with the same permissions, owner, and group. The names of the group and owner are transmitted along with the file rather than the group ID and User ID because user and group names might have different group ID and User ID values on different servers.

The grid must exist and you must run the grid routines as an authorized user from an authorized server. Wildcard characters in file names are not supported.

1. On the source server, set the GRIDCOPY_DIR configuration parameter to the location of the file to copy.
2. On the destination servers, set the GRIDCOPY_DIR configuration parameter to the location of the destination of the file to copy.
3. Run the ifx_grid_copy() procedure specifying the grid name, the name of the file to send, and, optionally, the file destination.

Example

Examples

Example 1: Copy a file to servers in a grid

The following example copies the file `$INFORMIXDIR/tmp/myfile` to the other nodes within grid grid1.

```
EXECUTE PROCEDURE ifx_grid_copy("grid1", "tmp/myfile")
```

Example 2: Copy a file to servers in a grid and change the name on the destination servers

In the following example, assume that the GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on the source server and on the destination server. The following example copies the file $INFORMIXDIR/tmp/bin/sales-010512.exe on the source server to $INFORMIXDIR/tmp/bin/sales.exe on all servers within the grid mygrid.

```
EXECUTE PROCEDURE ifx_grid_copy ("mygrid", "bin/sales-010512.exe", "bin/sales.exe");
```

Related reference

ifx_grid_copy() procedure on page 531

GRIDCOPY_DIR Configuration Parameter on page 518

Rerunning failed grid routines

You can rerun a grid routine that failed on one or more servers in the grid.

About this task

If a grid routine failed on one or more servers in the grid, you can run the cdr list grid command with the --nacks option to see the details of why it failed. If a server in the grid is offline or is not connected to the network, then a grid routine will be pending on that server and will be run when the server is reconnected to the grid.

In some cases, you should not rerun a failed routine, because the failure is expected. For example, if a server already has the database object that a grid routine is creating, then that routine fails on that server. If a command failed on all grid servers, you can run the original command again instead of running the ifx_grid_redo() procedure.

The grid must exist and you must run the grid routine as an authorized user from an authorized server.

To rerun a grid routine, run the ifx_grid_redo() procedure.

If you run the ifx_grid_redo() procedure without additional arguments besides the grid name, all routines that failed are re-attempted on all the servers on which they failed. You can specify on which server to rerun routines and which routines to rerun.

Example

Example

Suppose you have a grid, named **grid1**, that contains the servers **gserv_1** and **gserv_2**, which have a database named **db1**.

You create a dbspace named **dbsp2** on the server **gserv_1** and then create a table in that dbspace in a grid context with the following commands:

```
$ dbaccess db1 -
execute procedure ifx_grid_connect('grid1');
create table t100 (c1 int primary key) in dbsp2;
execute procedure ifx_grid_disconnect();
```

The cdr list grid command shows that the command failed on the server **gserv_2**:

```
$ cdr list grid grid1 --nack
Grid                     Node                     User
------------------------ ------------------------ ------------------------
grid1                    gserv_1*                 user1
                         gserv_2
Details for grid grid1

Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
NACK gserv_2 2011-02-24 09:27:45 SQLERR:-261 ISAMERR:-130
     Grid Apply Transaction Failure
```

The error indicates that the table could not be created because the specified dbspace does not exist.

You create a dbspace named **dbsp2** on the server **gserv_2** and run the ifx_grid_redo() procedure to rerun the original command on **gserv_2**:

```
$ dbaccess db1 -
execute procedure ifx_grid_redo('grid1');
```

The output of the cdr list grid command shows that the command succeeded on both servers:

```
$ cdr list grid grid1 -v
Grid                     Node                     User
------------------------ ------------------------ ------------------------
grid1                    gserv_1*                 user1
```

```
                         gserv_2
Details for grid grid1
...
Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
ACK gserv_1 2011-02-24 09:27:44
ACK gserv_2 2011-02-24 09:31:09
```

Related reference

ifx_grid_redo() procedure on page 536

cdr list grid on page 391

Connection management for client connections to participants in a grid

You can configure Connection Managers to route connection requests from clients to the replication servers of a grid.

Connection requests can be directed to replication servers based on Connection Manager service-level agreements (SLAs). You can configure Connection Manager SLAs to redirect connection requests based on various redirection policies. Connection Managers support the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

Related information

Connection management through the Connection Manager on page

Example of configuring connection management for a grid or replicate set on page

Example of setting up a replication system with a grid on page 125

Grid queries

If you have a table that is the same on multiple servers in a grid, but whose data is not replicated, you can run a grid query to return the consolidated data from the multiple servers.

For example, suppose that you have a chain of retail stores. Each store has a database with the same schema. The database contains tables for inventory, customer data, and sales transactions. You set up a grid because you want to replicate the inventory tables to a central server. You want the tables for sales transactions to be the same on every server, but you do not want to replicate all the sales transactions to the central server. You do, however, want a monthly report that shows the total sales per store. You run a grid query on the central server that aggregates the sales data for the last month for each store and returns results that are grouped by store.

To run a grid query, you include the GRID clause in the SELECT statement. The GRID clause specifies the grid, or subset of the grid, on which to run the query. The GRID clause has requirements and restrictions for the tables and other SQL constructs that you can include in the query.

Before you can run a grid query, you must define the table that you want to query as a grid table. If you use secure connections between your grid servers, you must configure secure connections on the grid server from which you want to run grid queries.

Planning for grid queries

Consider the following options when you plan grid queries.

Before you run a grid query, you can configure the following options for the queries:

- Whether to run the grid query on all the servers in the grid or a subset of grid servers. To define subsets of grid servers, create regions by running the cdr define region command. You can create as many grid regions as you need. Grid regions can overlap or be divided into smaller grid regions. A grid server can be a member of multiple grid regions.
- Whether to make all SELECT statements that are run in the current session run as grid queries by default. Run the SET ENVIRONMENT SELECT_GRID or the SET ENVIRONMENT SELECT_GRID_ALL statements to specify the grid or region name for every query. Leave the GRID clause out of SELECT statements.
- Whether to skip grid servers that are not available when you run the grid query. By default, the grid query runs only if all servers are available. Run the SET ENVIRONMENT GRID_NODE_SKIP ON statement to run the query on the available servers and skip the unavailable servers.

While you run a grid query, besides choosing the tables and the grid or region to include in the query, you can include the following options:

- Whether to return all qualifying rows, including duplicate rows. By default, grid queries return only unique rows. Include the ALL keyword in the GRID clause to return all rows.
- Whether to return information about which server the results are from. Include the ifx_node_id() or ifx_node_name() function to return a column that identifies the grid server from which each row originates. You can use the server ID or name to group the results.

After you run a grid query, you can find out which servers were skipped for a grid query, if the GRID_NODE_SKIP option was set to ON. Run the ifx_gridquery_skipped_node_count() and ifx_gridquery_skipped_nodes() functions to return the grid servers that were unavailable during the grid query.

Related reference

ifx_grid_connect() procedure on page 527

cdr define region on page 341

cdr delete region on page 373

cdr change gridtable on page 297

cdr remaster gridtable on page 431

ifx_node_id() function on page 543

ifx_node_name() function on page 544

ifx_gridquery_skipped_nodes() function on page 541

ifx_gridquery_skipped_node_count() function on page 542

ifx_grid_release() function on page 537

ifx_grid_remove() function on page 539

Related information

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID_NODE_SKIP session environment option on page

GRID clause on page

Defining tables for grid queries

Define the tables that you want to include in grid queries as grid tables.

Before you begin

The only prerequisite for defining a table as a grid table is that the table must have the same name, column names, and data types on multiple grid servers. However, the GRID clause has other restrictions and requirements for running grid queries.

About this task

You can include system catalog and **sysmaster** databases tables in grid queries without defining them as grid tables.

To define a table as a grid table, run the cdr change gridtable command. The cdr change gridtable command verifies that the tables have matching column names and data types across the grid.

Example

For example, the following command defines the **items**, **orders**, and **customer** tables in the **stores_demo** database for the grid named **grid1**:

```
cdr change gridtable --grid=grid1 --database=stores_demo --add items orders customer
```

What to do next

If you want to alter a grid table, you must run the alter operation through the grid. You cannot run a grid query on the table during an alter operation. After the alter operation is complete, the database server verifies that the table is consistent across grid servers.

Related reference

cdr change gridtable on page 297

cdr remaster gridtable on page 431

Related information

GRID clause on page

Configuring secure connections for grid queries

If the `sqlhosts` files on the grid servers include the `s=6` option, you must define alternate connections for grid queries. On the grid server from which you want to run grid queries, create a `grid.servers` file that lists the server group names and aliases for the other grid servers.

About this task

You do not need to encrypt the file. Authentication is done through normal authentication methods.

To configure secure connections for grid queries:

On the grid server from which you want to run grid queries, create a text file named `grid.servers` in the `INFORMIXDIR/etc` directory. List each grid server group name and alias on a separate line.

Example

For example, the following `sqlhosts` file for a grid uses the `s=6` option for secure connections:

```
#dbservers    nettype   hostname   servicename   options

g_ca_sf       group     -          -             i=100
san_francisco ontlitcp  computer1  sf_alt        g=g_ca_sf,s=6

g_ca_sj       group     -          -             i=200
san_jose      ontlitcp  computer2  sj_alt        g=g_ca_sj,s=6

g_ca_okl      group     -          -             i=300
oakland       ontlitcp  computer3  okl_alt       g=g_ca_okl,s=6

g_ca_yk       group     -          -             i=400
yreka         ontlitcp  computer4  yk_alt        g=g_ca_yk,s=6

g_ca_sac      group     -          -             i=500
sacramento    ontlitcp  computer5  sac_alt       g=g_ca_sac,s=6

g_ca_stk      group     -          -             i=600
stockton      ontlitcp  computer6  stk_alt       g=g_ca_stk,s=6
```

The corresponding `grid.servers` file has the following contents:

```
#group     alias
g_ca_sf    sf_alt
g_ca_sj    sj_alt
g_ca_okl   okl_alt
g_ca_yk    yk_alt
```

```
g_ca_sac   sac_alt
g_ca_stk   stk_alt
```

Related reference

ifx_grid_connect() procedure on page 527

ifx_grid_release() function on page 537

ifx_grid_remove() function on page 539

Related information

Configuring secure ports for connections between replication servers on page 59

GRID clause on page

Examples of grid queries

These examples show some of the options that you have when you run grid queries.

The following examples are based on the **stores_demo** database. A grid named **grid1** has eight servers, named **store1** through **store8**. The examples assume that you defined the **items**, **orders**, and **customer** tables as grid tables.

Example

Example 1: Return chunk information about grid servers

Suppose you want to know about the chunks on all your grid servers. You want to know the number of chunks, which dbspaces each chunk is in, the total size of each chunk, and the amount of free space in each chunk.

You run the following grid query to return chunk information for each grid server. The tables in the **sysmaster** database are grid tables by default.

```
database sysmaster;

SELECT ifx_node_name()::char(12) AS node, chknum, dbsnum, nfree, chksize
FROM syschunks GRID ALL 'grid1';
```

The grid query returns the following results:

```
node         chknum dbsnum       nfree     chksize

store1            1      1     1777275     2000000
store1            2      2        5025      100000
store1            3      3       24974      100000
store2            1      1     1775579     2000000
store2            2      2        5025      100000
store2            3      3       24974      100000
store3            1      1     1769260     2000000
store3            2      2        5025      100000
store3            3      3       24974      100000


. . .
```

Example

Example 2: Aggregate results by server and find skipped servers

Suppose you want a list of the orders by customer for each store in the grid named **grid1**. A store is represented by its grid server name. You want to return all results, including duplicate rows. You do not want the query to fail if any of the grid servers are unavailable, but you want to know which servers were skipped.

Before you run the grid query, you run the following statement to run the query on available grid servers and skip any unavailable grid servers:

```
SET ENVIRONMENT GRID_NODE_SKIP ON;
```

You run the following grid query to return the outstanding orders by customer for each store:

```
SELECT c.fname, c.lname, ifx_node_name() AS node
  SUM(i.total_price) AS tot_amt, SUM(i.quantity) AS tot_cnt
  FROM items i, orders o, customer c GRID ALL 'grid1'
  WHERE i.order_num = o.order_num
  AND o.customer_num = c.customer_num
  GROUP BY 1,2
  ORDER BY 2,1,3;
```

The grid query returns the following results:

```
fname    Alfred
lname    Grant
node     store1
tot_amt  $84.00
tot_cnt  2

fname    Alfred
lname    Grant
node     store2
tot_amt  $84.00
tot_cnt  4

. . .
```

You run the following statement to find how many grid servers were skipped:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();

2
```

Two servers were skipped. You run the ifx_gridquery_skipped_nodes() statement for each of the skipped servers:

```
EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

store5

EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

store8
```

Example

Example 3: Query a region of the grid

Suppose you want to know the total sales and number of sales per person for each store in Kansas. Kansas has two stores whose grid servers are named **store3** and **store4**. You want all queries during your database session to be run as grid queries for the Kansas stores.

You run the following command to define a grid region named **region1** that contains the servers **store3** and **store4**:

```
cdr define region --grid=grid1 region1 store3 store4
```

You run the following statement to set all SELECT statements during the session as grid queries for the region **region1**:

```
SET ENVIRONMENT SELECT_GRID_ALL region1
```

You run the following statement to return the total sales and number of sales per person for each store. The GRID clause is not necessary because you set the SELECT_GRID_ALL option.

```
SELECT fname[1,10], lname[1,10], ifx_node_id() AS storenum,
  SUM(quantity) AS tot_cnt, SUM(total_price) AS tot_amt
  FROM items i, orders o, customer c
  WHERE i.order_num = o.order_num
  AND o.customer_num = c.customer_num
  GROUP BY 2,1
  ORDER BY 2,1,3;
```

The query returns the following results:

```
fname      lname         storenum          tot_cnt          tot_amt

Alfred     Grant                3                8           $84.00
Alfred     Grant                4                6           $84.00
Marvin     Hanlon               3               12          $438.00
Marvin     Hanlon               4               10          $438.00
Anthony    Higgins              3               45         $1451.80
Anthony    Higgins              4               36         $1451.80
Roy        Jaeger               3               16         $1390.00
Roy        Jaeger               4               13         $1390.00
Fred       Jewell               3               16          $584.00
Fred       Jewell               4               13          $584.00
Frances    Keyes                3                4          $450.00
Frances    Keyes                4                3          $450.00


. . .
```

Example

Example 4: Use a grid query as a subquery

Suppose you want the total sales and number of sales for each customer across all stores. You use the same query that you use in example 2 as the subquery to return information by grid server. The main query aggregates the results of the subquery.

You run the following statement to return the total sales and number of sales per person:

```
SELECT fname, lname,
   SUM(tot_amt) AS amt_by_person, SUM(tot_cnt) AS tot_by_person
   FROM
```

```
       (
         SELECT c.fname, c.lname, ifx_node_name() AS node,
            SUM(i.total_price) AS tot_amt, SUM(i.quantity) AS tot_cnt
            FROM items i, orders o, customer c GRID ALL 'grid1'
            WHERE i.order_num = o.order_num
            AND o.customer_num = c.customer_num
            GROUP BY 1,2
        )
   GROUP BY fname, lname
   ORDER BY 2, 1;
```

The query returns the following results:

```
fname           lname              amt_by_person   tot_by_person

Alfred          Grant                    $336.00              20
Marvin          Hanlon                  $1752.00              40
Anthony         Higgins                 $5807.20             135
Roy             Jaeger                  $5560.00              50
Fred            Jewell                  $2336.00              50
Frances         Keyes                   $1800.00              10
Margaret        Lawson                  $1792.00             110

. . .
```

Related reference

ifx_grid_connect() procedure on page 527

cdr define region on page 341

cdr delete region on page 373

cdr change gridtable on page 297

cdr remaster gridtable on page 431

ifx_node_id() function on page 543

ifx_node_name() function on page 544

ifx_gridquery_skipped_nodes() function on page 541

ifx_gridquery_skipped_node_count() function on page 542

ifx_grid_release() function on page 537

ifx_grid_remove() function on page 539

Related information

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID_NODE_SKIP session environment option on page

GRID clause on page

Shard cluster setup

Sharding is a way to horizontally partition a single table across multiple database servers in a shard cluster. Enterprise Replication moves the data from the source server to the appropriate target server as specified by the sharding method. You query a sharded table as if the entire table is on the local server. You do not need to know where the data is. Queries that are performed on one shard server retrieve the relevant data from other servers in a shard cluster. Sharding reduces the index size on each shard server and distributes performance across hardware. You can add shard servers to the shard cluster as your data grows.

Prerequisites

Before you create a shard cluster, the following system must be in place:

- You must have an Enterprise Replication domain that is composed of two or more nodes.
- On one of the Enterprise Replication nodes, you must have a table or collection to shard that conforms to the following requirements:
    - The table must have a dedicated column or field for tracking row or document versions.
    - The table cannot include data types that are not supported in sharded queries.
    - The databases on all shard servers must have same locale type.

Shard cluster architecture

Shard servers are uniquely identified by the SHARD_ID configuration parameter that you must set on each shard server. Because shard servers have unique IDs, Enterprise Replication can efficiently communicate between shard servers:

- Client connections are multiplexed over a common pipe and authenticated only on the local shard server.
- Sharded queries are run in parallel on all shard servers and their high-availability secondary servers.
- For insert, update, and delete operations, if you set the USE_SHARDING session environment option, transactions use the two-phase commit protocol to move data to appropriate shard server. Otherwise, changes are moved to appropriate shard server using the eventually consistent model after the transaction is committed. For select operations, if you set the USE_SHARDING session environment option, queries are run on all shard servers in the cluster instead of on only the local database server.
- The consistency of the sharded table is enforced on all shard servers. Shard servers do not need to transfer table information between each other. Data definition language statements that you run on a sharded table are propagated to all shard servers.

Related information

JSON data sharding on page

Components supporting high availability and scalability on page

Creating a shard cluster

To create a shard cluster, prepare the shard servers and specify the sharding definition.

About this task

All shard servers must belong to the same Enterprise Replication domain.

To create a shard cluster:

1. On each shard server, set the SHARD_ID configuration parameter to a positive integer value that is unique in the shard cluster by running the following command:

   ```
   onmode -wf SHARD_ID=unique_positive_integer
   ```

   If the SHARD_ID configuration parameter is already set to a positive integer, you can change the value by editing the `onconfig` file and then restarting the database server. You can also set the SHARD_MEM configuration parameter to customize the number of memory pools that are used during shard queries.
2. On the shard server that contains the table to shard, run the cdr define shardCollection command.

What to do next

When applications connect to shard servers, enable sharded queries to run against data across all shard servers by setting the USE_SHARDING session environment variable:

```
SET ENVIRONMENT USE_SHARDING ON;
```

Related reference

cdr define shardCollection on page 361

SHARD_ID configuration parameter on page 519

Related information

onmode -wf, -wm: Dynamically change certain configuration parameters on page

USE_SHARDING session environment option on page

Shard cluster definitions

The definition for a shard cluster includes information about the shard servers, the data to shard, and the sharding method.

To run the cdr define shardCollection command, which creates a sharding definition for partitioning your table data, you must specify the following information:

- A name for the sharding definition
- The name of the database that contains the table that is being sharded
- The name of the user that owns the table that is being sharded
- The `sqlhosts` file group name for each database server in the shard cluster
- The column that is used as a shard key
- Which sharding method the database server uses for determining where rows are distributed to:

    - With consistent hash-based sharding, the data is automatically distributed between shard servers in a way that minimizes the data movement when you add or remove shard servers.
    - With hash-based sharding, the data is automatically divided between shard servers, but when you change the shard cluster, all data is redistributed.
    - With expression-based sharding, you specify how the data is divided between shard servers. You must also specify the shard server to receive the data that is outside the scope of the expression.
- How you want to distribute the data:
    - Insert rows on any shard server, replicate the rows to the appropriate shard server, and then delete duplicate rows from the original server. The delete method is the default method and is the same behavior as when you define sharding with MongoDB commands.
    - Insert rows on any shard server, replicate the rows to the appropriate shard server, but then keep duplicate rows on the original server. The keep method is similar to a data dissemination system.
    - Insert rows on the appropriate shard server and do not replicate rows. The informational method is useful if you want to query across multiple servers that have the same table, but you do not need to shard the data during loading. For example, you have a different database server for each of your three stores. The data from each store is always inserted in the appropriate server. You set up the sharding definition with an expression that matches database servers with their store identifiers. Then you can run sharded queries to aggregate data from all three stores.
- The table column or collection field for tracking row updates

Consistent hash-based sharding

When you create a consistent hash-based sharding definition, HCL Informix® uses a hash value of a specific column or field to distribute data to the servers of a shard cluster in a consistent pattern. When you add or remove a shard server, the consistent hashing algorithm redistributes a fraction of the data. You specify how many hashing partitions to create on each shard server. The default number of hashing partitions is three. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.

For example, the following command creates a consistent hashing index that has three partitions on each shard server:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

You can dynamically change the number of hashing partitions per shard server by running the cdr change shardCollection command.

Hash-based sharding

When you create a hash-based sharding definition, HCL Informix® uses a hash value of a specific column or field to distribute data to the servers of a shard cluster. When you add or remove a shard server, the hashing algorithm redistributes all the data.

For example, the following command creates a hashed index that is based on shard key values, and then the Enterprise Replication determines where rows with specific hashed index values are distributed to:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A g_shard_server_B g_shard_server_C g_shard_server_D
```

Expression-based sharding

When you create an expression-based sharding definition, HCL Informix® uses WHERE-clause syntax on a specific column or field to distributes data to the servers of a shard cluster.

For example, the following command sends rows with a shard-key value of `NV` to **g_shard_server_B**:

```
cdr define shardCollection collection_1 db_1:joe.clients
--type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('WA','OR','ID')"
   g_shard_server_B "IN ('CA','NV','UT','AZ')"
   g_shard_server_C "IN ('TX','OK','NM','AR','LA')"
   g_shard_server_D  REMAINDER
```

Sharding definitions must include the `REMAINDER` expression for rows or documents that have values that are not accounted for by the other expressions. For example, the previous sharding definition sends rows with a shard-key value of `'NY'` to **g_shard_server_D**.

Expressions that are used for sharding data cannot overlap. For example, a sharding definition that is created with the following command is not valid because rows or documents with shard key values `40` to `60` would be sent to both **g_shard_server_A** and **g_shard_server_B**.

```
cdr define shardCollection collection_1 db_1:joe.clients)
--type=delete --key=age --strategy=expression --versionCol=version
   g_shard_server_A "BETWEEN 0 AND 60"
   g_shard_server_B "BETWEEN 40 AND 100"
   g_shard_server_C  REMAINDER
```

Sharded queries

You can query a sharded table as if it is a single table on one database server. However, restrictions for distributed queries between database servers and restrictions specific to sharded queries apply.

When you run a sharded query, do not include server name qualifications for remote servers.

If the SHARD_ID configuration parameter is set to unique values on each shard server in the shard cluster, sharded queries are run in parallel on each shard server.

If you set the USE_SHARDING session environment option, insert, update, and delete operations on shard tables use the two-phase commit protocol. Otherwise, sharded insert, update, and delete operations follow the eventually consistency model where data is moved to the appropriate shard server after the transaction is committed.

If your shard servers have high-availability secondary servers, you can run sharded queries from the secondary servers.

Data types

A sharded query can return the following data types: non-opaque atomic built-in data types, LVARCHAR, Boolean, BSON, and JSON. Sharded queries cannot return distinct data types.

To run sharded queries on time series data in a **TimeSeries** data type, shard a virtual table that is based on the time series table.

Restrictions

You cannot include the following SQL syntax elements in a query that includes a sharded table:

- DataBlade API routines
- Java user-defined routines
- Triggers
- A FOR UPDATE clause in a SELECT statement

You cannot run an EXECUTE FUNCTION or EXECUTE PROCEDURE statement for a routine to operate on a sharded table.

You cannot run a statement that contains an update to a shard key that requires the row to move to another shard server. To update the shard key of a row, delete the row and then insert it with the new values.

You cannot shard data in an XA environment.

Sharded Table Joins

HCL Informix® supports joining between multiple sharded tables with parallel execution. However, such parallel joins between two shard tables are allowed ONLY when the following conditions are met:

- both the tables must have the joining column as key
- both the tables must have exactly the same strategy defined on the key
- both the tables must have exact partitioning conditions defined using the key
- both the tables must have same set of the participating nodes
- only equi-joins are allowed in case of other non-expression strategies

All the table filters on the sharded tables will be pushed to their respective participants

When the shard-join is rejected due to any restriction, Informix server will attempt a fall back mechanism.

Shard join fallback is enabled using the following command:

SET ENVIRONMENT SHARDJOIN_FALLBACK ON. For more information, see SHARDJOIN_FALLBACK session environment option on page .

Performance tips

You can improve the speed of sharded queries by customizing how shared memory for sharded queries is allocated. You can control shared memory allocation by setting the SHARD_MEM configuration parameter on each shard server.

If your sharded queries frequently include joins to another table, replicate that table to all the shard servers to improve query performance.

If your sharded queries included stored routines as a filter, define the routines on all the shard servers. Queries run faster when the data is filtered on each shard server before being returned.

If the SHARD_ID configuration parameter is set on all shard servers, the shard servers use server multiplexer group (SMX) connections. You can reduce latency between shard servers by increasing the number of pipes that are used for the SMX connections. Set the SMX_NUMPIPES configuration parameter to the number of pipes.

Related reference

SHARD_MEM configuration parameter

Shard cluster management and monitoring

You can scale out a shard cluster by adding new shard servers. You can also change the shard cluster's definition to change where rows or documents are distributed to.

Modifying a shard-cluster and adding or removing shard servers

To modify the servers in a shard cluster, run the cdr change shardCollection command on one of the shard servers. The cdr change shardCollection command performs the following actions:

1. A new sharding definition is created.
2. Using the new sharding definition, existing data is distributed across the shard servers.
3. The original sharding definition is deleted.

Note: You can add new servers, remove existing servers, or modify the sharding definition, but you cannot change the type of sharding definition. A hash-based sharding definition cannot change to an expression-based sharding definition, and an expression-based sharding definition cannot change to a hash-based sharding definition.

The following example shows how to add capacity to a shard cluster that uses a hash-based sharding definition. The original shard cluster was defined with the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=identifier --strategy=hash --versionCol=version
   g_shard_server_1
   g_shard_server_2
```

Run the following command to add **g_shard_server_3** to the shard cluster:

```
cdr change shardCollection collection_1 --add g_shard_server_3
```

The following example shows how to add a new region-specific server to a shard cluster that uses an expression-based sharding definition. The original shard cluster was created with the following command:

```
cdr define shardCollection collection_2 db_2:john.clients
   --type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_1 "IN ('WA','OR')"
   g_shard_server_2 "IN ('CA','NV')"
   g_shard_server_3  remainder
```

Run the following command to add **g_shard_server_4** to the shard cluster:

```
cdr change shardCollection collection_2 --add
   g_shard_server_4 "IN ('UT','ID')"
```

A new sharding definition is created. Any rows or documents that have a shard key of **UT** or **ID** are moved from **g_shard_server_3** to **g_shard_server_4**, and all future inserts are distributed according to the new sharding definition.

You can remove servers from a shard cluster using the --drop option or entirely replace a sharding definition with the --replace option.

Monitoring a shard cluster

To see the current definition for a shard cluster, you can run the cdr list shardCollection command on one of the shard servers, and specify the definition's name. For example:

```
cdr list shardCollection my_collection
```

To see information on the shard cache, run the onstat -g shard command on one of the shard servers.

Stopping data distribution and deleting a sharding definition

To stop data distribution and delete the sharding definition, run the cdr delete shardCollection on one of the shard servers, and specify the definition's name. For example:

```
cdr delete shardCollection my_collection
```

Related reference

cdr change shardCollection on page 304

cdr delete shardCollection on page 380

cdr list shardCollection on page 405

Related information

onstat -g shard command: Print information about the shard cache on page

Shard edge server

Shard edge server simplifies the administration of large shard cluster when shard cluster is only used for shard query functionality.

If you do not require automatic data partitioning, then you can designate one or more servers as shard edge servers so that you only need to define Enterprise replication at the shard coordinator server. Shard edge servers are standard Informix servers and you cannot start shard query from shard edge server.

Use the SHARD_EDGE_NODE configuration parameter to enable shard server as a edge server. For more information, see SHARD_EDGE_NODE configuration parameter.

Example

This example shows how to configure shard cluster between three servers: server1, server2 and server3. We will make server1 as our shard coordinator and server2 and server3 as shard edge servers.

1. Configure SQLHOSTS

SQLHOSTS file for server1:

```
group1 group - - i=1
server1 onsoctcp host1 10000 g=group1
group2 group - - i=2
server2 onsoctcp host2 10000 g=group2
group3 group - - i=3
server3 onsoctcp host3 10000 g=group3
```

Note: Even though we do not require Enterprise Replication defined for server2 and server3, server1 sqlhosts file need to be configured with group entries for server2 and server3.

SQLHOSTS file for server2:

```
SQLHOSTS file for server2:
server2 onsoctcp host2 10000
server1 onsoctcp host1 10000
```

SQLHOSTS file for server3:

```
SQLHOSTS file for server3:
server3 onsoctcp host3 10000
server1 onsoctcp host1 10000
```

2. Establish trusted host relationship between server1 and server2, and between server1 and server3

3. Define Enterprise replication for server1

```
cdr define server –connect server -I group1
```

4. Update server2 and server3 config file to set SHARD_EDGE_NODE config value to 1

5. For parallel shard query function, make sure to set unique value to SHARD_ID config parameter at server1, server2 and server3

6. Create shard definition one or more tables to run shard queries

```
cdr define shardCollection –connect server1 customer_shard stores_demo:usr1.sales_bson --type
 informational_noer --key
  bson_value_lvarchar(sales_data, 'amount') --strategy hash group group group
```

Managing Replication Servers and Replicates

These topics cover how to manage your Enterprise Replication system, including managing replication servers, replicates and participants, replicate sets, templates, replication server network connections, and resynchronizing data, and performing alter operations on replicated tables.

Managing Replication Servers

About this task

You manage replication servers with the **cdr** commands.

The *state* of the server refers to the relationship between the source server and the target server. To determine the current state of the server, use the **cdr list server *server_name*** command. For more information about the possible server states, see cdr list server on page 402.

Note: When switching a server to administration mode to perform administrative tasks, be aware that any Enterprise Replication on the server will be started (or continue to run normally if already started). In this situation data on which you might be relying may change as other users modify it, and concurrency problems may arise as others access the same data. To avoid this problem, launch the server using the **oninit -Dj** command; if the server is already running, use the **cdr stop** command to shut down any currently running replications.

Related reference

Set configuration parameters for replication on page 72

Modify server attributes

You modify replication server attributes by running the cdr modify server command.

You can change the following attributes of the server:

- Idle timeout
- Whether Aborted Transaction Spooling (ATS) files or Row Information Spooling (RIS) files are generated
- Location of the directory for the ATS or RIS files
- The format of the ATS files: text, XML, or both
- The mode of all the participants on the server: primary, receive-only, or send-only
- The mode of all the participants on the server: primary or receive-only

Related reference

cdr modify server on page 421

Related information

Defining Replication Servers on page 93

Dynamically Modifying Configuration Parameters for a Replication Server

You can alter the settings for Enterprise Replication configuration parameters and environment variables on a replication server while replication is active.

About this task

Use the following commands to dynamically update values of most Enterprise Replication configuration parameters:

cdr add onconfig

Adds a value. This option is available only for configuration parameters and environment variables that allow multiple values.

cdr change onconfig

Replaces the existing value. This option is available for all Enterprise Replication configuration parameters and environment variables.

cdr remove onconfig

Removes a specific value. This option is available only for configuration parameters and environment variables that allow multiple values.

The commands change configuration parameters in the `onconfig` file. To update environment variables, use the CDR_ENV configuration parameter.

To dynamically update the value of the CDR_DELAY_PURGE_DTC configuration parameter, use the onmode -wf command.

The following table shows which changes are valid for Enterprise Replication configuration parameters.

Table 14. Options for dynamically updating Enterprise Replication configuration parameters

Configuration Parameter	cdr add onconfig	cdr change onconfig	cdr remove onconfig
CDR_APPLY	No	No	No
CDR_DBSPACE	No	Yes	No
CDR_DSLOCKWAIT	No	Yes	No
CDR_ENV CDR_ALARMS	No	No	No
CDR_ENV CDR_LOGDELTA	No	Yes	No
CDR_ENV CDR_PERFLOG	No	Yes	No

Table 14. Options for dynamically updating Enterprise Replication configuration parameters (continued)

Configuration Parameter	cdr add onconfig	cdr change onconfig	cdr remove onconfig
CDR_ENV CDR_RMSCALEFACT	No	Yes	No
CDR_ENV CDR_ROUTER	No	Yes	No
CDR_ENV CDRSITES_731	Yes	Yes	Yes
CDR_ENV CDRSITES_92X	Yes	Yes	Yes
CDR_ENV CDRSITES_10X	Yes	Yes	Yes
CDR_EVALTHREADS	No	Yes	No
CDR_LOG_LAG_ACTION	Yes	Yes	Yes
CDR_LOG_STAGING_MAXSIZE	Yes	Yes	Yes
CDR_MAC_DYNAMIC_LOGS	No	Yes	No
CDR_NIFCOMPRESS	No	Yes	No
CDR_QDATA_SBSPACE	Yes	Yes	Yes
CDR_QUEUEMEM	No	Yes	No
CDR_SERIAL	No	Yes	No
CDR_SUPPRESS_ATSRISWARN	Yes	Yes	Yes
ENCRYPT_CDR	No	Yes	No
ENCRYPT_CIPHERS	No	Yes	No
ENCRYPT_MAC	Yes	Yes	Yes
ENCRYPT_MACFILE	Yes	Yes	Yes
ENCRYPT_SWITCH	No	Yes	No

You can view the setting of Enterprise Replication configuration parameters and environment variables with the onstat -g cdr config command.

Related reference

onstat -g cdr config: Print ER settings on page 551

cdr add onconfig on page 289

cdr change onconfig on page 298

cdr remove onconfig on page 434

Enterprise Replication configuration parameter and environment variable reference on page 498

Viewing Replication Server Attributes

About this task

After you define a server for replication, you can view information about the server using the **cdr list server** command. If you do not specify the name of a defined server on the command line, Enterprise Replication lists all the servers that are visible to the current server. If you specify a server name, Enterprise Replication displays information about the current server, including server ID, server state, and attributes.

For more information, see cdr list server on page 402.

Connect to another replication server

By default, when you view information about a server, Enterprise Replication connects to the global catalog of the database server specified by the **INFORMIXSERVER** environment variable. You can connect to the global catalog of another database server by using the --connect option.

About this task

For example, to connect to the global catalog of the database server **idaho**, enter: `cdr list server --connect=idaho`

Related information

Enterprise Replication Terminology on page 3

Connect Option on page 256

Temporarily stopping replication on a server

You can temporarily stop replication on a server to perform maintenance tasks in several different ways.

About this task

You can stop Enterprise Replication on a server by shutting down the database server. Replication begins again when you restart the database server.

However, you might want to temporarily stop the Enterprise Replication threads without stopping the database server.

You can temporarily stop replication by running the **cdr stop** command. The stopped server does not capture data to be replicated. Other replication servers in the domain continue to queue replicated data for the stopped server in their send queues. Replication threads remain stopped (even if the database server is stopped and restarted) until you run the **cdr start** command. When you restart replication on the server, it receives and applies the replicated data from the other replication servers. However, if replication is stopped for long enough, the replay position on the logical log on the stopped server can be overrun and the send queues on the active replication servers can fill up. If either of these situations happens, you must synchronize the server that was stopped.

If your replicates use time stamp or delete wins conflict resolution rules, you should temporarily stop replication on the server by using the cdr disable server command. Disabling a replication server is also appropriate if you do not have enough disk space to avoid overrunning the replay position. Replication servers do not queue replicated transactions for the disabled replication server, nor does the disabled replication server queue its transactions. Therefore, you must synchronize the replication server that was disabled after you enable replication on it by using the cdr check replicateset command. However, because information about deleted rows on the disabled replication server is saved in delete tables, you can take advantage of a time stamp repair.

Related reference

cdr stop on page 465

cdr disable server on page 383

Restarting Replication on a Server

You can restart replication after Enterprise Replication was temporarily stopped.

About this task

If replication was stopped by the cdr disable server command, you can restart it by running the cdr check replicateset command with the --repair and the --enable options or by running the cdr enable server command. If you use the cdr enable server command, you must subsequently synchronize the server.

If replication stopped due to an error, you can restart replication by shutting down and restarting the database server or by running the cdr start command.

If replication was stopped by the cdr stop command, restart replication by running the cdr start command.

When you run the cdr start command, Enterprise Replication resumes evaluating the logical logs at the replay position (where Enterprise Replication stopped evaluating the logical log when the server was stopped). If the replay position was overwritten in the logical log, replication cannot restart and event alarm 75 is raised. In this situation, run the **cdr cleanstart** command to restart Enterprise Replication and then synchronize the data.

Related reference

cdr start on page 443

cdr enable server on page 388

Suspending Replication for a Server

About this task

If you do not want to completely shut down the Enterprise Replication threads, you can suspend replication of data to the server using the cdr suspend server command. When replication is suspended to the server, the source server queues replicated data but suspends delivery of replicated data to the target server. Note that this command does not affect the network connection to the suspended server. The source server continues to send other messages, such as acknowledgment and control messages.

For example, to suspend replication of data to the server group **g_papeete** from the server group **g_raratonga**, enter: `cdr suspend server g_papeete g_raratonga`

To suspend replication to **g_papeete** from all servers in the enterprise, enter:

```
cdr suspend server g_papeete
```

Important: When you suspend replication on a server, you must ensure that the send queues on the other Enterprise Replication servers participating in replication do not fill.

For more information, see cdr suspend server on page 473.

Resuming a Suspended Replication Server

About this task

To resume replication to a suspended server, use the **cdr resume server** command, specifying which server you want to resume. When you resume the server, the queued data is delivered.

For example, to resume replication to the **g_papeete** server group, enter:

```
cdr resume server g_papeete
```

For more information, see cdr resume server on page 442.

Deleting a Replication Server

You can remove Enterprise Replication from a database server and then remove the database server from an Enterprise Replication domain.

About this task

Run the cdr delete server command two times to remove Enterprise Replication from a database server, and then remove the database server from an Enterprise Replication domain. The first time, run the command on the server you want to remove Enterprise Replication from. The second time, connect to a different server in the Enterprise Replication domain and run the command, specifying the server you ran the first command on.

To remove Enterprise Replication from an inactive database server, use the cdr delete server command with the --force option.

To restart Enterprise Replication on a disabled database server, define the server again with the cdr define server command and then synchronize data. Row history is deleted when a server has Enterprise Replication removed, so the history is not recoverable if Enterprise Replication is restarted.

Warning: Do not delete an Enterprise Replication server and then immediately recreate it with the same name. If you recreate the objects before the operation finishes propagating to the other Enterprise Replication database servers in the domain, failures might occur in the Enterprise Replication system at the time of the operation or later.

Important: If you are creating a replicate to replace the one you deleted, use the cdr check queue --qname=ctrlq command to make sure that the delete operation propagated to all the servers.

Example

Examples

To remove Enterprise Replication from local database server **reynolds**, and then remove database server **reynolds** from the Enterprise Replication domain it shares with database server **stimpson**, run the following commands:

```
cdr delete server reynolds
cdr delete server --connect=stimpson reynolds
```

The first command removes Enterprise Replication from the local database server, **reynolds**. The second command connects to database server **stimpson**, which is another server in the Enterprise Replication domain, and then removes database server **reynolds** from the shared domain.

Related reference

cdr delete server on page 377

cdr check queue on page 310

Managing Replicates

You can perform the following tasks on existing replicates:

- Modify replicate attributes or participants
- View replicate properties and state
- Change the state of a replicate (whether replication is being performed)
- Delete a replicate

Modify replicates

You can modify replicates while replication is active to add or remove participants, or to change some replicate attributes.

To change other attributes of replicates, you create a new replicate and then delete the original replicate.

Adding or Deleting Participants

About this task

To be useful, a replicate must include at least two participants. You can define a replicate that has fewer than two participants, but before you can use that replicate, you must add more participants.

To add a participant to an existing replicate, use the **cdr change replicate --add** command. For example, to add two participants to the **sales_data** replicate, enter:

```
cdr change replicate --add sales_data \
   "db1@hawaii:jane.table1" "select * from table1" \
   "db2@maui:john.table2" "select * from table2"
```

To delete a participant from the replicate, use the **cdr change replicate --delete** command.

For example, to delete these two participants from the replicate, enter:

```
cdr change replicate --delete sales_data \
   "db1@hawaii:jane.table1" "db2@maui:john.table2"
```

For more information, see cdr change replicate on page 299.

Change replicate attributes

You can change many replicate attributes by running the cdr modify replicate command.

You can change the following attributes of a replicate:

- Conflict-resolution rules and scope
- Replication frequency
- Error logging
- Replication of full rows or only changed columns
- Database triggers
- Participant type
- Code set conversion
- Serial processing
- Replication of unchanged large objects

You cannot change the conflict resolution from ignore to a non-ignore option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-ignore conflict resolution option to ignore.

For example, to change the replication frequency for the **sales_data** replicate to every Sunday at noon, enter:

```
cdr modify replicate sales_data Sunday.12:00
```

Related reference

cdr modify replicate on page 415

Changing the replication key of a replicate

You can change the replication key of a replicate from the primary key, a unique index or constraint, or the ERKEY shadow columns to another unique index or constraint on your table.

To change the replication key of a replicate:

1. Define a new replicate by running the cdr define replicate command. Include the --key option to specify the new replication key. Do not include the --erkey option.
2. Start the new replicate by running the cdr start replicate command.
3. Stop the original replicate by running the cdr stop replicate command.
4. Delete the original replicate by running the cdr delete replicate command.

Related reference

cdr define replicate on page 342

Related information

Unique key for replication on page 25

Changing or re-creating primary key columns on page 194

Viewing Replicate Properties

About this task

After you define a replicate, you can view the properties of the replicate using the **cdr list replicate** command. If you do not specify the name of a defined replicate on the command line, Enterprise Replication lists detailed information on all the replicates defined on the current server. If you use the **brief** option, Enterprise Replication lists participant information about all the replicates. If you specify a replicate name, Enterprise Replication displays participant information about the replicate.

For information about this command, see cdr list replicate on page 395.

Starting a Replicate

About this task

When you define a replicate, the replicate does not begin until you explicitly change its state to *active*. When a replicate is active, Enterprise Replication captures data from the logical log and transmits it to the active participants. At least two participants must be active for data replication to occur.

Important: You cannot start replicates that have no participants.

To change the replicate state to active, use the **cdr start replicate** command. For example, to start the replicate **sales_data** on the servers **server1** and **server23**, enter:

```
sales_data server1 server23
```

This command causes **server1** and **server23** to start sending data for the **sales_data** replicate.

If you omit the server names, this command starts the replicate on all servers that are included in that replicate.

When you start a replicate, you can choose to perform an initial data synchronization, as described in Initially Synchronizing Data Among Database Servers on page 120.

Warning: Run the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.

When replication is active on an instance, you may need to double the amount of lock resources, to accommodate transactions on replicated tables.

If a replicate belongs to an exclusive replicate set, you must start the replicate set to which the replicate belongs. For more information, see Starting a Replicate on page 169.

For more information, see cdr start replicate on page 445.

Stopping a Replicate

You can temporarily stop replication for administrative purposes.

About this task

To stop the replicate, use the **cdr stop replicate** command. This command changes the replicate state to *inactive* and deletes any data in the send queue for that replicate. When a replicate is inactive, Enterprise Replication does not transmit or process any database changes.

In general, you should only stop replication when no replication activity is likely to occur for that table or on the advice of IBM® Software Support. If database activity does occur while replication is stopped for a prolonged period of time, the replay position in the logical log might be overrun. If a message that the replay position is overrun appears in the message log, you must resynchronize the data on the replication servers. For more information on resynchronizing data, see Resynchronizing Data among Replication Servers on page 177.

You cannot stop replicates that have no participants.

For example, to stop the **sales_data** replicate on the servers **server1** and **server23**, enter:

```
cdr stop replicate sales_data server1 server23
```

This command causes **server1** and **server23** to purge any data in the send queue for the **sales_data** replicate and stops sending data for that replicate. Any servers not listed on the command line continue to capture and send data for the **sales_data** replicate (even to **server1** and **server23**).

If you omit the server names, this command stops the replicate on all servers that are included in that replicate.

If a replicate belongs to an exclusive replicate set, you must stop the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets on page 118 and Stopping a Replicate Set on page 174.

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see cdr stop replicate on page 468.

Suspending a Replicate

About this task

If you do not want to completely halt all processing for a replicate, you can suspend a replicate using the **cdr suspend replicate** command. When a replicate is in a suspended state, the replicate captures and accumulates changes to the source database, but does not transmit the captured data to the target database.

Warning: Enterprise Replication does not support referential integrity if a replicate is suspended. Instead, you should suspend a server. For more information, see Suspending Replication for a Server on page 165.

For example, to suspend the **sales_data** replicate, enter:

```
cdr suspend replicate sales_data
```

If a replicate belongs to an exclusive replicate set, you must suspend the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets on page 118 and Suspending a Replicate Set on page 174.

For more information, see cdr suspend replicate on page 471.

Resuming a Suspended Replicate

About this task

To return the state of a suspended replicate to active, use the **cdr resume replicate** command. For example:

```
cdr resume replicate sales_data
```

If a replicate belongs to an exclusive replicate set, you must resume the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets on page 118 and Resuming a Replicate Set on page 175.

For more information, see cdr resume replicate on page 440.

Deleting a Replicate

To delete a replicate from the global catalog, use the cdr delete replicate command.

About this task

When you delete a replicate, Enterprise Replication purges all replication data for the replicate from the send queue at all participating database servers.

For example, to delete the **sales_data** replicate from the global catalog, enter:

```
cdr delete replicate sales_data
```

Warning: Avoid deleting a replicate and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later. For more information, see Asynchronous propagation conflicts on page 19.

Important: If you are creating a replicate to replace the one you deleted, use the cdr check queue --qname=ctrlq command to make sure that the delete operation propagated to all the servers.

Related reference

cdr check queue on page 310

cdr delete replicate on page 374

Managing Replicate Sets

When you create a replicate set, you can manage the replicates that belong to that set together or individually. If the replicate set is exclusive, you can only manage the individual replicates as part of the set.

Performing an operation on a replicate set (except **cdr delete replicateset**) is equivalent to performing the operation on each replicate in the replicate set individually.

For more information, see Managing Replicates on page 167.

Connection management for client connections to participants in a replicate set

You can configure Connection Managers to route connection requests from clients to the replication servers of a replicate set.

Connection requests can be directed to replication servers based on Connection Manager service-level agreements (SLAs). You can configure Connection Manager SLAs to redirect connection requests based on various redirection policies. Connection Managers support the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

Related information

Connection management through the Connection Manager on page

Example of configuring connection management for a grid or replicate set on page

Modifying Replicate Sets

You can modify replicate sets in two ways:

- Add or Delete Replicates
- Change Replication Frequency

Adding or Deleting Replicates From a Replicate Set

About this task

To add a replicate to an existing replicate set, use the command **cdr change replicateset --add**. For example, to add two replicates to **sales_set**, enter:

```
cdr change replicateset --add sales_set sales_kauai \
   sales_moorea
```

The state of the replicate when you add it to a replicate set depends on the type of replicate set:

- For a non-exclusive replicate set, the state of the new replicate remains as it was when you added it to the set. To bring all the replicates in the non-exclusive set to the same state, use one of the commands described in Managing Replicate Sets on page 172.
- For an exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.

To delete a replicate from the replicate set, use **cdr change replicate --delete**.

For example, to delete the two replicates, **sales_kauai** and **sales_moorea**, from the replicate set, enter:

```
cdr change replicateset --delete sales_set sales_kauai\
   sales_moorea
```

When you add or remove a replicate from an exclusive replicate set that is suspended or that is defined with a frequency interval, Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate. For more information, see Suspending a Replicate Set on page 174 and Frequency Options on page 287.

For more information, see cdr change replicateset on page 302.

Changing Replication Frequency For the Replicate Set

About this task

You can change the replication frequency for the replicates in an exclusive or non-exclusive replicate set using the **cdr modify replicateset** command. For more information, see Specifying Replication Frequency on page 104.

For example, to change the replication frequency for each of the replicates in the **sales_set** to every Monday at midnight, enter:

```
cdr modify replicateset sales_set Monday.24:00
```

For more information, see cdr change replicateset on page 302.

Viewing Replicate Sets

About this task

To view the properties of the replicate set, use the **cdr list replicateset** command. The **cdr list replicateset** command displays the replicate set name and a list of the replicates that are members of the set. To find out more about each replicate in the replicate set, see Viewing Replicate Properties on page 169.

For more information, see cdr list replicateset on page 400.

Starting a Replicate Set

About this task

To change the state of all the replicates in the replicate set to active, use the **cdr start replicateset** command. For example, to start the replicate set **sales_set**, enter:

```
set sales_set
```

When you start a replicate set, you can choose to perform an initial data synchronization, as described in Initially Synchronizing Data Among Database Servers on page 120.

Warning: Run the **cdr start replicateset** command on an idle system (when no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement after you successfully start the replicate.

For more information, see cdr start replicateset on page 448 and cdr start replicate on page 445.

Stopping a Replicate Set

About this task

To stop the replicates in the replicate set, use the **cdr stop replicateset** command. This command changes the state of all the replicates in the set to inactive.

For example, to stop the **sales_set** replicate set, enter:

```
cdr stop replicateset sales_set
```

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see cdr stop replicateset on page 469 and cdr stop replicate on page 468.

Suspending a Replicate Set

About this task

If you do not want to completely halt all processing for the replicates in a replicate set, you can suspend the replicates in the set using the **cdr suspend replicateset** command.

For example, to suspend the **sales_set** replicate set, enter:

```
cdr suspend replicateset sales_set
```

For more information, see cdr suspend replicateset on page 472 and cdr suspend replicate on page 471.

Related reference

cdr change replicateset on page 302

Resuming a Replicate Set

About this task

To return the suspended replicates in the replicate set to active, use the **cdr resume replicateset** command. For example:

```
cdr resume replicateset sales_set
```

For more information, see cdr resume replicateset on page 441 and cdr resume replicate on page 440.

Deleting a Replicate Set

To delete a replicate set, use the cdr delete replicateset command.

About this task

Tip: When you delete a replicate set, Enterprise Replication does not delete the replicates that are members of the replicate set. The replicates remain in the state they were in when the set was deleted.

For example, you can connect to the default database server specified by the **INFORMIXSERVER** environment variable and delete the **sales_set** replicate set by using running the following command:

```
cdr delete replicateset sales_set
```

Warning: Avoid deleting a replicate set and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later. For more information, see Asynchronous propagation conflicts on page 19.

Important: If you are creating a replicate to replace the one you deleted, use the cdr check queue --qname=ctrlq command to make sure that the delete operation propagated to all the servers.

Related reference

cdr check queue on page 310

cdr delete replicateset on page 376

Managing Templates

About this task

You can use the **cdr list template** and **cdr delete template** commands to view information about your templates and to clean up obsolete templates. The commands are described in detail, including examples and sample output, in The cdr utility on page 253.

You cannot update a template. To modify a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

Viewing Template Definitions

About this task

Use the **cdr list template** command to view detailed information about the template and the servers, databases and tables for which the template defines replication.

Deleting Templates

About this task

Use the **cdr delete template** command to delete any templates that you no longer want to use to set up replication. The command also deletes any replicate sets associated with the template which exist if the template has been realized.

Important: Deleting a template does not delete replicates that have been created by realizing a template.

Managing Replication Server Network Connections

This section explains how you can view network connections status, drop network connections, and reestablish dropped network connections.

Viewing Network Connection Status

About this task

To determine the current status of the network connection to each of the servers participating in replication, use the **cdr list server** command and look at the STATUS column of the output.

For more information, see cdr list server on page 402.

Dropping the Network Connection

About this task

To drop the Enterprise Replication network connection for a server, use the **cdr disconnect server** command. When you drop the connection, Enterprise Replication continues to function and queue transactions. For example, to disconnect the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr disconnect server g_papeete
```

Warning: When you disconnect a server from Enterprise Replication, you must ensure that the send queues on **all** other Enterprise Replication servers participating in replication do not fill.

For more information, see cdr disconnect server on page 385.

Reestablishing the Network Connection

About this task

To reestablish a dropped network connection, use the **cdr connect server** command.

For example, to reestablish the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr connect server g_papeete
```

The following conditions can cause reestablishing a network connection to fail:

- A network outage
- A server is offline
- The cdr stop, cdr disconnect server, or cdr delete server commands were run on a server
- The system clock times on the servers differ by more than 900 seconds

For more information, see cdr connect server on page 337.

Resynchronizing Data among Replication Servers

If replication has failed for some reason and data is not synchronized, there are different ways to correct data mismatches between replicated tables.

The following table compares each of the methods. All methods except manual table unloading and reloading can be performed while replication is active.

Table 15. Resynchronization methods

Method	Description
Direct synchronization	• Replicates all rows from the specified reference server to all specified target servers for a replicate or replicate set. • Runs as a foreground process by default, but can run as a background process. • Populates tables in a new participant. • Quickly synchronizes significantly inconsistent tables when used with the TRUNCATE statement.
Checking consistency and then repairing inconsistent rows	• Compares all rows from the specified target servers with the rows on the reference server, prepares a consistency report, and optionally repairs inconsistent rows. • Runs as a foreground process by default, but can run as a background process.
ATS or RIS file repairs	• Used to repair rows that other synchronization methods could not repair. • Repairs a single transaction at a time. • Replicates or replication server must have been configured with the ATS or RIS option.
Manual table unloading and reloading	• Manual process of unloading the target table, copying the reference table, and then loading the reference table into the target database. • Requires that replication be suspended.

Related reference

cdr stop on page 465

cdr stop replicate on page 468

Related information

Repair and Initial Data Synchronization on page 8

Performing Direct Synchronization

Direct synchronization replicates every row in the specified replicate or replicate set from the reference server to all the specified target servers. You can use direct synchronization to populate a new target server, or an existing target server that has become severely inconsistent.

Before you begin

- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing direct synchronization.
- The replicate must not be in a suspended or stopped state during direct synchronization.
- The replicate must not be set up for time based replication.

About this task

You can synchronize a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

Important: Running direct synchronization can consume a large amount of space in your log files. Ensure you have sufficient space before running this command.

To perform direct synchronization, use the **cdr sync replicate** or **cdr sync replicateset** command.

You can monitor the progress of a synchronization operation with the **cdr stats sync** command if you provide a progress report task name in the **cdr sync replicate** or **cdr sync replicateset** command.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the --name option so that you can monitor the operation with the cdr stats sync command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the --process option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If direct synchronization cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

Related reference

cdr sync replicate on page 476

cdr sync replicateset on page 481

cdr stats sync on page 461

Related information

Repairing Failed Transactions with ATS and RIS Files on page 187

Synchronizing Significantly Inconsistent Tables

If your target tables are significantly inconsistent, you can speed the synchronization process by truncating the target tables before you perform direct synchronization.

About this task

When you truncate a table by using the TRUNCATE statement, you remove all rows from the table while replication is active. After the tables on the target servers are empty, direct synchronization efficiently applies data from the source server to the target servers.

If you use the TRUNCATE statement on the supertable in a hierarchy, by default, rows in all the subtables are deleted as well. You can use the ONLY keyword to limit the truncate operation to the supertable. For more information on the TRUNCATE statement, see the *HCL® Informix® Guide to SQL: Syntax*.

To synchronize tables in conjunction with truncation:

1. Run the TRUNCATE statement on the tables to be synchronized on the target servers.
2. Run the **cdr sync replicate** or **cdr sync replicateset** command.

Results

For the syntax of these commands, see cdr sync replicate on page 476 and cdr sync replicateset on page 481.

Checking Consistency and Repairing Inconsistent Rows

A consistency check compares the data between a reference server and one or more target servers and then generates a report that describes any inconsistencies. You can choose to repair inconsistent rows during a consistency check.

Before you begin

The following conditions apply when you check consistency:

- Running a consistency check can consume a large amount of space in your log files. Ensure you have sufficient space before checking consistency.
- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing consistency checking and repair.
- The replicate must not be in a suspended or stopped state during consistency checking.
- The replicate must not be set up for time based replication.

About this task

You can perform a consistency check and optional synchronization on a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

To perform a consistency check, use the **cdr check replicate** or **cdr check replicateset** command. Use the **--repair** option to repair the inconsistent rows. A consistency report is displayed for your review.

You can monitor the progress of a consistency check with the **cdr stats check** command if you provide a progress report task name in the **cdr check replicate** or **cdr check replicateset** command.

You can run a consistency check as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the --name option so that you can monitor the check with the cdr stats check command. You can also view the command and its results in the command_history table in the **sysadmin** database. If you use the --background option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

Results

If synchronization during a consistency check cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

Related reference

cdr check replicate on page 313

cdr check replicateset on page 325

cdr stats check on page 457

Related information

Repairing Failed Transactions with ATS and RIS Files on page 187

Indexing the ifx_replcheck Column on page 183

Interpreting the Consistency Report

The consistency report displays information about differences in replicated data within the replicate or replicate set.

Inconsistencies listed in the consistency report do not necessarily indicate a failure of replication. Data on different database servers is inconsistent while replicated transactions are in progress. For example, the following consistency report indicates that two rows are missing on the server **g_serv2**:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         0         0

WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The missing rows could be in the process of being replicated from **g_serv1** to **g_serv2**.

If you choose to repair inconsistent rows during a consistency check, the report shows the condition of the replicate at the time of the check, plus the actions taken to make the replicate consistent. For example, the following report shows two missing rows on **g_serv2** and that two rows were replicated from **g_serv1** to correct this inconsistency:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 65         0         2         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The warning indicates that inconsistencies were discovered.

The report indicates whether the replicate became consistent after the repair process. In this example, the `Validation of repaired rows failed.` message indicates that the replicate is not consistent. This might occur because some replicated transactions were still being replicated. Use the **--inprogress** option to extend the validation time.

The verbose form of the consistency report also displays the differing values for each inconsistent row.

For more information about the contents of the consistency report, see cdr check replicate on page 313.

Related reference

cdr check replicate on page 313

Increase the speed of consistency checking

You can increase the speed of checking the consistency of replicates or replicate sets with the cdr check replicate or cdr check replicateset commands in several ways.

About this task

To increase the speed of consistency checking of replicate sets by checking the member replicates in parallel, use the --process option to set the number of parallel processes equal to the number of replicates.

To increase the speed of consistency checking by limiting the amount of data that is checked, use one or more of the following options:

- Skip the checking of large objects with the --skipLOB option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the --since option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- When checking a replicate, you can check a subset of the data with the **--where** option.

If you have large tables, you can index the **ifx_replcheck** shadow column.

Related reference

cdr check replicateset on page 325

cdr check replicate on page 313

Indexing the ifx_replcheck Column

You can index the **ifx_replcheck** shadow column to increase the speed of consistency checking.

Before you begin

If you have a large replicated table, you can add the **ifx_replcheck** shadow column and then create a new unique index on that column and the existing replication key columns. The index on the **ifx_replcheck** shadow column allows the database server to determine whether rows in different tables have different values without comparing the values in those rows. You must create the index on the table in each database server that participates in the replicate.

Before you can create an index on the **ifx_replcheck** shadow column and the replication key columns, you must prepare the replicated table by adding the **ifx_replcheck** shadow column. You can add the **ifx_replcheck** shadow column when you create the table with the WITH REPLCHECK clause, or you can alter an existing table to add the **ifx_replcheck** shadow column with the ADD REPLCHECK clause.

About this task

You can create the index while replication is active.

To index the **ifx_replcheck** shadow column, create a unique index based on the existing replication key columns and the **ifx_replcheck** column. The **ifx_replcheck** shadow column must be the last column in the index.

Example

For example, the following statement creates an index on a table named customer on the primary key column **id** and **ifx_replcheck**:

```
CREATE UNIQUE INDEX customer_index ON  customer(id, ifx_replcheck);
```

Related reference

cdr check replicate on page 313

cdr check replicateset on page 325

Related information

Checking Consistency and Repairing Inconsistent Rows on page 180

Preparing Tables for a Consistency Check Index on page 78

Repair inconsistencies by time stamp

You can repair inconsistencies based on the latest time stamps among the participants instead of specifying a master server.

If your replicates use the time stamp or delete wins conflict resolution rule, you can repair inconsistencies between the participants based on the latest time stamp on any participant. If you run a time stamp repair, you do not specify a master server whose data is considered correct and to which all the other participants are matched.

To ensure that a time stamp repair is accurate, follow these guidelines:

- When you need to temporarily stop replication on a server, disable it with the cdr disable server command instead of stopping it with cdr stop command.
- If you are using the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter on all replication servers to the maximum age of modifications to rows that are being actively updated.

To run a time stamp repair, use the cdr check replicate or cdr check replicateset command with the **--repair** and **--timestamp** options. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option.

If a time stamp repair finds an extra row on any participant, the result depends on the conflict resolution rule and the last transaction for that row:

- If the conflict rule is time stamp and the most recent time stamp for the row is a delete transaction, the row will be deleted on all servers.
- If the conflict rule is time stamp and a participant has a deleted row but the most recent time stamp for that row is an update transaction, the updated row is replicated to all servers.
- If the conflict rule is delete wins and any participant has deleted that row, the row is deleted from all servers, regardless of any later update transactions.

If a time stamp repair finds mismatched rows on different servers, then the most recent update transaction for that row is replicated to the other server.

Related reference

CDR_DELAY_PURGE_DTC configuration parameter on page 500

Related information

Time stamp conflict resolution rule on page 42

Delete wins conflict resolution rule on page 48

Repairing inconsistencies while enabling a replication server

If a replication server is in disabled mode, you can enable it and repair inconsistencies with the cdr check replicateset command.

Before you begin

The server must have been put in disabled mode with the cdr disable server command.

About this task

To enable a disabled server and synchronize it, run the cdr check replicateset command with the --repair and --enable options.

By default, the enable process times out after 128 seconds if the disabled replication server cannot be enabled and repaired during that time. You can specify a shorter time out period by setting the --timeout option to a value less than or equal to 60 seconds.

To repair all replicate sets on the disabled server, also include the --allrepl option and omit the --replset option.

Related reference

cdr check replicateset on page 325

Implementing a custom checksum function

You can implement a custom checksum function for consistency checking if you do not want to use the checksum function that is included with the database server.

About this task

The `$INFORMIXDIR/extend/checksum` directory contains sample checksum function code and registration statements.

To implement a custom checksum function:

1. Using the `idschecksum.c` file in the `$INFORMIXDIR/demo/checksum` directory as a template, write a C language function that creates a checksum.
   Overload the function for each of the supported data types.
2. Compile the function code into a shared object file.
3. Save a copy of the shared object file in the `$INFORMIXDIR/extend/checksum` directory on all replication servers.
4. To register the function:
   a. Modify the `idschecksum.sql` file in the `$INFORMIXDIR/demo/checksum` directory to include the name of your function.
   b. Run the SQL statements on each replication server.

What to do next

Specify your checksum function name with the --checksum option when you run the cdr check replicate or cdr check replicateset command.

If the cdr check replicate or cdr check replicateset command fails with return code 172, your checksum function is not installed and registered on all replication servers.

Rules for custom checksum functions

The `idschecksum.c` file in the `$INFORMIXDIR/demo/checksum` directory contains code that you can use for your checksum functions. You can replace the checksum generation portion of the code with your custom code.

A checksum function summarizes the data in a replicated row. During consistency checking, the checksum values of corresponding rows on different replication servers are compared to determine whether the rows are consistent.

A checksum function runs recursively through every column in the replicated table to generate a checksum value for a replicated row. A checksum value is generated for the last column in the table. The checksum value is used to calculate the checksum value for the previous column, and so on. The checksum value that is calculated for the first column in the table is based on the accumulated checksum value of all the other columns.

Custom checksum functions must conform to the following rules:

- The first parameter of the function is the data type of a column.
- The second parameter of the function is an integer that is the checksum value of the previous column.
- The function returns an integer.
- The function is a DBA function.
- The function attributes include NOT VARIANT, HANDLESNULLS, and PARALLELIZABLE.

You must register a checksum function for each of the following data types, regardless of whether the data types are used in your replicated tables. All other data types are ignored by checksum functions.

- BLOB
- BYTE
- CLOB
- DATE
- DATETIME YEAR TO FRACTION
- DATETIME YEAR TO MONTH
- DECIMAL
- FLOAT
- INT8
- INTEGER
- LIST
- LVARCHAR (includes all character data types)
- MONEY
- MULTISET
- REAL
- ROW
- SET
- SMALLINT
- TEXT

However, if you do not want to create a checksum value for certain data types, you can provide non-operative function definitions. For example, you might not want to create checksum values for BLOB columns. The following statement registers a checksum function for the BLOB data type that returns the previous checksum value instead of calculating an accumulated checksum value:

```
CREATE DBA FUNCTION ercheck_checksum(p1 blob, p2 integer)
      RETURNS integer;
RETURN p2;
END FUNCTION;
```

Repairing Failed Transactions with ATS and RIS Files

You can repair failed or inconsistent transactions using an ATS or RIS file if you defined the replicate or replication server with the **-ats** or **-ris** option and the ATS or RIS files are being generated in text format.

About this task

A repair using an ATS or RIS file repairs the rows associated with the single transaction that is recorded in the specified ATS or RIS file. To apply repairs based on an ATS or RIS file, use the **cdr repair** command.

Note: The cdr repair command is not supported for replicates that are defined with the `--UTF8=y` option. For replicates that are defined with the `--UTF8=y` option, use the cdr check replicate --repair or cdr check replicateset --repair command to repair data.

The **cdr repair** command processes one ATS or RIS file each time you specify the command. The following table shows how failed operations are handled.

Failed Operation	Action Taken
Delete	Delete on the target server
Insert or Update	• If the row is found on the source server, does an update • If the row is not found on the source server, but is found on the target server, does a delete on the target server. If the row is not found on either server, performs no action.

Each operation is displayed to stderr, unless you use the **-quiet** option with the **cdr repair** command. You can preview the operations without performing them by using the **-check** option with the **cdr repair** command.

Related information

Failed Transaction (ATS and RIS) Files on page 199

The cdr utility on page 253

Performing Direct Synchronization on page 178

Checking Consistency and Repairing Inconsistent Rows on page 180

Repair and Initial Data Synchronization on page 8

Resynchronize data manually

Manual resynchronization involves replacing the inconsistent table in the target database with a copy of the correct table from the reference database.

Important: Manual resynchronization is not recommended for repairing your replicated tables because you must suspend replication to avoid producing further inconsistencies.

The following example shows how to manually resynchronize two replication database servers.

To synchronize the replication server **g_papeete** with the server **g_raratonga**:

1. Suspend replication to the replication server group **g_papeete.**

   See Suspending Replication for a Server on page 165.

2. Unload the table from the server group **g_raratonga.**

   See Load and unload data on page 82.

3. Load the table on **g_papeete** and specify BEGIN WORK WITHOUT REPLICATION.

   See Load and unload data on page 82 and Blocking Replication on page 76.

4. Resume replication to **g_papeete**.

   See Resuming a Suspended Replication Server on page 166.

Alter, rename, or truncate operations during replication

When Enterprise Replication is active and data replication is in progress, you can perform many types of alter, rename, or truncate operations on replicated tables and databases.

Most of the supported operations do not require any special steps when performed on replicated tables or databases; some, however, do require special steps. None of the supported alter, rename, or truncate operations are replicated. You must perform these operations on each replicate participant.

You can run the alter, rename, and truncate operations that are listed in the following table on active, replicated tables or databases without performing extra steps.

Table 16. Requirements for operations on replicated tables

Operation	Requirements
Add or drop default values and SQL checks	None

Table 16. Requirements for operations on replicated tables (continued)

Operation	Requirements
Add or drop fragments	Requires mastered replicate to be defined
Add or drop unique, distinct, and foreign keys	None
Alter the locking granularity	None
Alter the next extent size	None
Change an existing fragment expression on an existing dbspace	Requires mastered replicate to be defined
Convert a fragmented table to a non-fragmented table	Requires mastered replicate to be defined
Convert a non-fragmented table to a fragmented table	Requires mastered replicate to be defined
Convert from one fragmentation strategy to another	Requires mastered replicate to be defined
Create a clustered index	Requires mastered replicate to be defined
Modify the data type of a replicated column	Requires mastered replicate to be defined
Modify the data type of a replicated column in a multiple-column replication key	Requires mastered replicate to be defined
Move a fragment expression from one dbspace to another dbspace	Requires mastered replicate to be defined
Move a non-fragmented table from one dbspace to another dbspace	Requires mastered replicate to be defined
Recluster an existing index	Requires mastered replicate to be defined
Rename a database	None
Rename a replicated column	Requires non-strict mastered replicate to be defined
Rename a table	Requires non-strict mastered replicate to be defined
Truncate a replicated table	Requires mastered replicate to be defined

You can perform the following alter operations on active, replicated tables, but you must perform extra steps, which are described in following sections:

- Add a column to a replicated table
- Remove a column from replication
- Attach a fragment to a replicated table
- Change or recreate a replication key

Enterprise Replication uses shadow replicates to manage alter operations on replicated tables without causing any interruption to replication. By using shadow replicates, the replicate participants SELECT clause can be modified while replication is active. For example, a new column can be brought into the replicate definition, an existing replicated column can be removed from the replicate definition and the data type or size of a replicated column can be changed without interrupting replication. See Defining Shadow Replicates on page 103 for more information about shadow replicates.

Before altering a replicated table, ensure that you have sufficient log space allocated for long transactions, a sufficient number of locks available, and sufficient space available for the queue sbspace.

When you issue a command to alter a replicated table, Enterprise Replication places the table in *alter* mode before performing the alter operation. Alter mode is a state in which only DDL (data-definition language) and SELECT operations are allowed but DML (data-manipulation language) operations are not allowed. After the transaction that initiated the alter operation completes, Enterprise Replication unsets alter mode. Any schema changes are automatically applied to any delete tables.

The following restrictions apply when you use alter operations on replicated tables.

- Enterprise Replication must be in an active state, unless you are only adding or dropping check constraints and default values.
- Tables must have a master replicate defined.
- The DROP TABLE statement is not supported.

Recommendation: If you need to perform more than one alter operation, enclose them in a single transaction so that alter mode only needs to be set and unset one time.

For a list of common alter operation problems and how to solve them, see Troubleshooting Tips for Alter Operations on page 218.

Related reference

cdr alter on page 290

Related information

SQL statements and replication on page 21

Altering multiple tables in a replicate set

You can alter multiple replicated tables for replicates that belong to the same replicate set and then remaster those tables as a group.

About this task

Instead of remastering the replicates individually for each table that you alter, you create a derived replicate set that contains only the replicates that must be remastered. You do not specify the names of the replicates. The server identifies which replicates have tables that must be remastered and adds them to the derived set for you. After you remaster and synchronize the derived replicate set, you delete it.

To alter replicated tables in a replicate set:

1. Run the ALTER operation on the tables on all replication servers.
   If the replicate set is included in a grid, you can alter the tables on one server and propagate the changes to all other servers.
2. Create a derived replicate set by running the cdr define replicateset command with the --needRemaster option.
3. Remaster the tables in the derived replicate set by running the cdr remaster replicateset command.
   The replicate definitions are updated in the global catalogs of the replication servers.
4. Synchronize the derived replicate set by running the cdr check replicateset with the --repair option or by running the cdr sync replicateset command.
5. Drop the derived replicate set by running the cdr delete replicateset command.

Related reference

Example of rolling out schema changes in a grid on page 128

cdr define replicateset on page 355

cdr check replicateset on page 325

cdr sync replicateset on page 481

cdr remaster replicateset on page 432

cdr delete replicateset on page 376

Adding a Replicated Column

About this task

You can alter a replicated table to add a new column to be replicated. The replicate must be a master replicate.

To add a new replicated column

1. Use the ALTER TABLE statement to add the column to the replicated table at all participating nodes.
2. Remaster the replicate to include the newly added column in the replicate definition, as described in Remastering a Replicate on page 195.

Removing replicated columns

You can alter replicated tables to remove columns from replication.

Before you begin

The replicates must be master replicates.

About this task

To remove one or more replicated columns from one or more replicates, run the cdr remaster command with the --remove option. You specify the database, table, and column names instead of the replicate names.

What to do next

After you remove columns from replication, you can drop the columns.

Related reference

cdr remaster on page 428

Related information

Adding an existing replicate to a grid replicate set by altering a table on page 135

Modifying the data type or size of a replicated column

You can modify the size or type of a replicated column for all basic data types and for the BOOLEAN and LVARCHAR extended types. Modifying the data type or size of columns of other extended types is not supported. The replicate must be a master replicate.

About this task

When you modify a replicated column, do not insert data into the modified column that does not fit into the old column definition until all participants are altered, because the data might be truncated or data conversion to and from the master dictionary format to the local dictionary format might fail. Enterprise Replication handles the data type mismatch by having the source server convert data that is in the local dictionary format to the master dictionary format, and the target server convert data from the master dictionary format to the local dictionary format. If Enterprise Replication detects a mismatch in data type or size between the master replicate definition and the local table definition, a warning is printed in the log file.

If Enterprise Replication is not able to convert the replicated row data into the master dictionary format on the source server while queuing replicated data into the send queue, the replicate is stopped for the local participant. If the replicate is stopped, you must correct the problem and then restart the replicate from the local participant with the --syncdatasource option. If the correction is to delete the problematic row data, delete the row by running the BEGIN WORK WITHOUT REPLICATION statement. Otherwise, the deleted row is moved from the replicated table to the associated delete table, which might cause problems for the subsequent alter operation on the replicated table.

If Enterprise Replication cannot convert row data from the master dictionary format to local table dictionary format at the target server after receiving replicated data, the replicated transaction is spooled to ATS and RIS files. For example, if you modify a SMALLINT column to an INTEGER column, make sure that you do not insert data that is too large for the SMALLINT data type until the alter operation is performed at all replicate participants, and remastering is performed so that the master dictionary reflects the INTEGER data type.

Important: While modifying a replicated column, sometimes it is possible that the alter operation on the base table succeeds, but the delete table modification might fail when Enterprise Replication unsets alter mode. If the delete modification fails, you see a message similar to the following in the server message log file:

```
CDRGC: cannot populate data into the new delete table
SQL error=-1226, ISAM error=0
```

This situation can happen while modifying a replicated column from a data type larger in length or size to a data type smaller in length or size, for example, from an INTEGER column to a SMALLINT column, and if the delete table has data which cannot fit in the new type column.

To avoid this situation, do not convert between data types that cause data truncation or produce cases where data cannot fit into the new type. If the above situation has already occurred, carefully update or delete the problematic rows from the delete table and attempt to unset alter mode manually by using the **cdr alter** command. If you cannot resolve the problem, contact IBM® Software Support.

To modify a replicated column:

1. Issue the alter command to modify the replicated column.
2. Perform the alter operation at all the replicate participants.
3. Optionally remaster the replicate to update the column definition in the replicate definition, as described in Remastering a Replicate on page 195.

Results

After an alter operation, the master dictionary no longer matches the replicated table dictionary. Because data transfer is always done in master dictionary format, data conversion between the local dictionary format and the master dictionary format is performed. Data conversion can slow the performance of your replication system. The remastering process changes the master dictionary to match the altered replicated table dictionary. Therefore, after remastering, data conversion is not necessary.

Replication keys have special considerations. For more information, see Changing or re-creating primary key columns on page 194.

Changing the Name of a Replicated Column, Table, or Database

About this task

You can change the name of a replicated column, table, or database while replication is active. The replicate must be a master replicate.

To change the name of a replicated column, table, or database, run the SQL statement RENAME COLUMN, RENAME TABLE, or RENAME DATABASE on all participants in the replicate.

Related information

RENAME TABLE statement on page

RENAME COLUMN statement on page

RENAME DATABASE statement on page

Changing or re-creating primary key columns

You can change or re-create the primary key columns definition of a replicated table while replication is active.

About this task

You can change the primary key columns without restriction if either of the following conditions are true:

- The table uses ERKEY shadow columns or another unique index or constraint as the replication key.
- The primary key contains multiple columns. The column modification implicitly re-creates the primary key.

To change a primary key column if the primary key is a single column, enclose the primary key column modification and the primary key recreation operations in a single transaction. If you frequently update a primary key that is a single column, consider changing the replication key to another unique index or constraint.

To drop and re-create a primary key:

1. Set alter mode by running the cdr alter on command.
2. Drop the primary key columns.
3. Create the new primary key columns.
4. Unset alter mode by running the cdr alter off command.

Related information

SQL statements and replication on page 21

Changing the replication key of a replicate on page 168

Attaching a New Fragment to a Replicated Table

You can attach a new fragment to a replication table while replication is active.

About this task

Enterprise Replication cannot automatically set alter mode for this operation because of an SQL restriction that requires attaching a fragment to be performed in multiple steps.

To attach a new fragment to a replicated table:

1. Set alter mode on the replicate by running the cdr alter on command.
2. Drop the replication key of the table.
3. Attach the new fragment.
4. Re-create the replication key.
5. Unset alter mode by running the cdr alter off command.

Related information

Preparing tables without primary keys on page 79

Remastering a Replicate

You must remaster a replicate if you add a replicated column, drop a replicated column, or change a classic replicate into a mastered replicate. If you modify a replicated column, you can remaster, but remastering is not mandatory.

About this task

To redefine an existing master replicate that is defined with name verification, or turn an existing classic replicate into a master replicate, run the cdr remaster command.

If the master replicate does not include name verification, you manually remaster the replicate.

Related reference

cdr remaster on page 428

Remastering replicates without name verification

You manually remaster replicates if the participants to not have matching column names and replicate has name verification turned off by the --name=n option of the cdr define replicate command.

To manually remaster a replicate:

1. Use the cdr define replicate command to create a shadow replicate with the same attributes as the primary replicate and with the --mirrors option, but with a SELECT statement that is correct for the table after the alter operation. The SELECT statement can include newly added columns or omit newly dropped columns.
2. Use the cdr swap shadow command to exchange the existing primary replicate and the newly created shadow replicate.

Results

While performing the cdr swap shadow operation, Enterprise Replication stores the BEGIN WORK position of the last known transaction sent to the grouper as a *swap log position* for the current swap operation. Any transaction that is begun before the swap log position uses the original replicate definition. Any transaction that is begun after the swap log position uses the new replicate definition.

The old replicate definition is deleted automatically after the replicate definition is no longer required by Enterprise Replication.

Related reference

cdr swap shadow on page 475

Recapture replicated transactions

If you want a transaction to continue to be replicated after it reaches the target replication servers, you can use the ifx_set_erstate() procedure.

By default, when Enterprise Replication reads the logical logs to capture transactions, replicated transactions are ignored. For example, if a transaction is replicated from **serv1** to **serv2**, that transaction is not captured for replication on **serv2** because it has already been replicated. Replication stops when transactions reach target servers, but you can configure a transaction to be recaptured and continue to be replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Example

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named low_inventory() that creates a low inventory report exists on all replications servers. The following example creates a new procedure named xqt_low_inventory() that enables replication for the low_inventory() procedure, and then runs the low_inventory() procedure:

```
CREATE PROCEDURE xqt_low_inventory()
 DEFINE curstate integer;
 EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
 EXECUTE PROCEDURE ifx_set_erstate(1);
 EXECUTE PROCEDURE low_inventory();
 EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
```

The following events occur in this procedure:

1. The xqt_low_inventory() procedure defines a data variable called **curstate** to hold the Enterprise Replication state information.
2. The ifx_get_erstate() function obtains the Enterprise Replication state and stores it in the **curstate** variable. The ifx_set_state() procedure enables replication.
3. The low_inventory() procedure is run.
4. The replication state is reset back to its original value.

When a transaction runs the xqt_low_inventory() procedure, the execution of the procedure is replicated to all replication servers and the result of the low_inventory() procedure is then replicated like any normal updating activity.

Related reference

ifx_set_erstate() procedure on page 546

ifx_get_erstate() function on page 526

Monitor and troubleshooting Enterprise Replication

You can monitor and diagnose problems with the Enterprise Replication system by using several different methods, depending on your needs.

You can monitor the status of Enterprise Replication servers in the following ways:

- Use the cdr view command. Specify one or more subcommands, depending on what information you want to monitor.
- Use SQL queries on the system monitoring tables.
- Run onstat commands to view local server information.
- Run the cdr check queue --qname=cntrlq command to determine whether the operation is finished propagating to all servers.
- Run the DBINFO('cdrsession') function to determine if a session thread is performing an Enterprise Replication apply or sync operation.

Set the ALARMPROGRAM script to capture event alarms for the following situations:

- Enterprise Replication errors
- The Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files
- Dropped connections between replication servers
- Replication state changes caused by Enterprise Replication commands, if state change event alarms are enabled

Related reference

cdr view on page 486

SMI Tables for Enterprise Replication Reference on page 574

onstat -g commands for Enterprise Replication on page 547

Enterprise Replication Event Alarms on page 221

cdr check queue on page 310

cdr define server on page 357

cdr modify server on page 421

Related information

Setting Up Failed Transaction Logging on page 105

DBINFO Function on page

Solve Replication Processing Problems

Diagnose, monitor, and solve possible problems that can occur while Enterprise Replication is running.

You should understand the typical behavior of your Enterprise Replication system. There are many factors that contribute to the performance and other behaviors, including: hardware configuration, network load and speed, type of replication, and number of replicated transactions.

Use the **cdr view** command or the SMI tables to understand the typical behavior of your system, establish benchmarks, and track trends. Deviations from typical behavior do not necessarily indicate a problem. For example, transactions might take longer to replicate during peak usage times or during end-of-month processing.

The following table describes some replication processing problems that might occur.

Table 17. Potential Replication Problems and Solutions

Problem	How to diagnose	How to solve
Enterprise Replication is not running	• Run the **cdr view state** command • Query the **syscdr_state** SMI table • Examine event alarms captured by the alarm program	Start replication with the **cdr start** command.
One or more Enterprise Replication servers are not running or connected to the network	• Run the **cdr view servers** command • Run the **cdr view nif** command • Query the **syscdr_nif** SMI table • Examine event alarms captured by the alarm program	Start the database server or fix the connection problem.
Replicated transactions failed	Determine if there are ATS or RIS files: • Look at the ATS and RIS directories on the local server for the existence of ATS or RIS files • Run the **cdr view atsdir risdir** command to see the number of ATS and RIS files for each server • Query the **syscdr_atsdir** or **syscdr_risdir** SMI table for a specific server • Examine event alarms captured by the alarm program	Run one of the following commands: • cdr repair • cdr check replicate --repair • cdr check replicateset --repair See cdr repair on page 435, cdr check replicate on page 313, and cdr check replicateset on page 325.

Table 17. Potential Replication Problems and Solutions (continued)

Problem	How to diagnose	How to solve
Transactions are spooling to disk	Determine how much spool memory is being used: • Run the **cdr view profile** command to see the status of all queues on all servers • Run the **cdr view sendq** command to see the status of the send queue on all servers • Run the **cdr view rcv** command to see the status of the receive queue on all servers	See Increasing the Sizes or Numbers of Storage Spaces on page 216.
Potential log wrap situation	Determine how many log pages must be used before Enterprise Replication reacts a potential log wrap situation: • Run the **cdr view ddr** command to see the number of unused log pages for all servers • Query the **syscdr_ddr** SMI table to see the number of unused log pages for a specific server	See Handle potential log wrapping on page 214.

If you do need to call IBM® Software Support, find the version of the database server that is running Enterprise Replication with the **cdr -V** command.

Failed Transaction (ATS and RIS) Files

Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files can be generated when replicated transactions fail.

You can use the ATS and RIS files to identify problems or as input to the **cdr repair** command or custom utilities that extract or reapply the aborted rows.

When ATS or RIS file generation is enabled for a replicate, all failed replication transactions are recorded in ATS or RIS files. Each ATS file contains all the information pertinent to a single failed transaction, while each RIS file contains information about a single failed row. If a replicated transaction fails for any reason (constraint violation, duplication, and so forth), all the buffers in the replication message that compose the transaction are written to a local file.

ATS file generation occurs if the entire transaction is aborted. Transactions defined with row scope that have aborted rows but are successfully committed on the target tables are not logged. All rows that fail conflict resolution for a transaction that has row scope defined are also written to the RIS file, if RIS is enabled.

RIS files can contain the following types of information:

- Individual aborted row errors
- Replication exceptions (such as when a row is converted by Enterprise Replication from insert to update, or from update to insert, and so forth)
- Special SPL routine return codes, as defined by the application (if an SPL routine is called to resolve a conflict)

In some cases, such as with long transactions, the database server itself aborts transactions. In these cases, Enterprise Replication does *not* generate an ATS or RIS file.

ATS and RIS files can be generated under the following circumstances:

- ATS or RIS generation is enabled for a replicate, the replicate uses a conflict resolution rule other than ignore or always-apply, and a conflict is detected on a target server.
- Under some error conditions, ATS or RIS files can be generated on a source server, regardless if ATS or RIS generation is enabled or the conflict resolution rule.

When an ATS or RIS file is generated, an event alarm with a class ID for 48 is also generated. You can use event alarms to send notifications to a database administrator.

Related reference

cdr view on page 486

CDR_DISABLE_SPOOL Environment Variable on page 521

cdr define replicate on page 342

Related information

Creating ATS and RIS directories on page 71

Repairing Failed Transactions with ATS and RIS Files on page 187

Conflict Resolution Scope on page 50

Enabling ATS and RIS File Generation

You can enable the generation of ATS and RIS files when you define a replicate.

About this task

Failed transactions are not automatically recorded in ATS and RIS files. You can choose to generate either ATS or RIS files, or both.

You should create a separate directory to store ATS and RIS files. If you do not create a separate directory and specify it when you define the replication server, Enterprise Replication stores the ATS and RIS files in the `/tmp` directory on UNIX™ and the `%INFORMIXDIR%\tmp` directory on Windows™.

To collect ATS and RIS information

1. Create a directory for Enterprise Replication to store ATS and RIS files. You can create two directories if you want to generate both types of file and store them in separate directories.

   Choose from:

   - If you are using primary-target replication, create the directory on the target system.
   - If you are using update-anywhere replication and have a conflict resolution rule other than ignore or always-apply enabled, create the directory on all participating replication systems.
2. When you define or modify a replication server, specify the location of the ATS and RIS directory by using the --ats and --ris options of the cdr define server command or the cdr modify server command.
3. When you define or modify a replicate, specify that ATS and RIS file generation is enabled by using the --ats and --ris options of the cdr define replicate command or the cdr modify replicate command.

Related reference

cdr define server on page 357

cdr define replicate on page 342

cdr modify server on page 421

cdr modify replicate on page 415

Related information

Creating ATS and RIS directories on page 71

ATS and RIS File Names

Each ATS and RIS file has a unique name based on the conditions under which it was generated.

The following table provides the naming convention for ATS and RIS files:

```
type.target.source.threadID.timestamp.sequence.extension
```

Table 18. ATS and RIS file naming conventions

Name	Description
type	The format of the file: `ats` or `ris`.
target	The name of the database server receiving this replicate transaction.
source	The name of the database server that originated the transaction.
threadID	The identifier of the thread that processed this transaction.
timestamp	The value of the internal time stamp at the time that this ATS or RIS file was generated.
sequence	A unique integer, incremented each time an ATS or RIS file is generated.
extension	The file type. No extension indicates a text file; `xml` indicates an XML file.

The naming convention ensures that all ATS and RIS file names that are generated are unique. However, when an ATS or RIS file is opened for writing, any previous file contents are overwritten. (Enterprise Replication does not append to a spool file; if a name collision does occur with an existing file, the original contents of the file are lost.)

The default delimiter for the *timestamp* portion of text file names is a colon (`:`) on UNIX™ and a period (`.`) on Windows™. You can define the delimiter between the hour, minute, and second values with the CDR_ATSRISNAME_DELIM environment variable. XML files always use a period (`.`) delimiter between the hour, minute, and second values.

The following is an example of a name of an ATS file in text format on UNIX™ for a transaction sent by server **g_amsterdam** to server **g_beijing**:

```
ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6
```

The following is an example of the same ATS file name in XML format:

```
ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6.xml
```

The following is an example of a similar RIS file name in XML format:

```
ris.g_beijing.g_amsterdam.D_2.000529_23.27.16.5.xml
```

Related reference

CDR_ATSRISNAME_DELIM Environment Variable on page 520

ATS and RIS File Formats

You can choose to generate ATS and RIS files in text format, XML format, or both formats.

The format of ATS and RIS files is part of the server definition that you create with the **cdr define server** command:

Text (Default)

ATS and RIS files are generated as text files that Enterprise Replication can process during a repair operation. Text format is useful if you intend to use the **cdr repair** command to repair inconsistencies.

XML

ATS and RIS files are generated as XML files that you can use if you write your own custom repair scripts. You cannot use ATS or RIS files in XML format with the **cdr repair** command.

Both

ATS and RIS files are generated in both text and XML format so that you can choose how to process failed transactions.

Enterprise Replication raises event alarms when ATS and RIS files are generated regardless of format.

XML File Format

The information in ATS and RIS files that are in XML format is organized in specific XML tags.

The XML format uses an XML schema that is stored in the **INFORMIXDIR/etc** directory.

Data in XML files uses the UTF-8 encoding format.

Columns that appear empty could contain a null value or an empty string. The XML format differentiates between null data and empty strings by setting the isNull="true" attribute of the COLUMN tag for null data.

Data Types That are Not Shown

The values of the following data types are not shown in XML files:

- Smart large objects
- Simple large objects
- User-defined data types

For these data types, the following attributes are set for the COLUMN tag:

- isLOBorUDT="true"
- dataExists="false"

Special Symbols

The following symbols are replaced if they exist in row data:

- `<` is replaced by `lt;`
- `>` is replaced by `>`
- is replaced by `amp;`
- `"` is replaced by `quot;`
- `'` is replaced by `apos;`

Example

Example

The following example shows an ATS file displaying a transaction with two failed insert operations. The third column in each row contains a data type that is not shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <ATS version="1">
    <TRANSACTION RISFile="/tmp/ris.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.3.xml"
                 generateRISFile="true" processedRows="2">
       <SOURCE id="20" name="g_cdr_ol_2" commitTime="2008-04-11T14:08:57"/>
       <TARGET id="30" name="g_cdr_ol_3" receiveTime="2008-04-11T14:08:57"/>
       <MESSAGE>All rows in a transaction defined with row scope were rejected</MESSAGE>
     </TRANSACTION>
     <ATSROWS>
       <ATSROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
              operation="Insert">
         <REPLICATED>
```

```
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                          cdrTimeString="2008-04-11T14:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">261</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
           </DATA>
         </REPLICATED>
       </ATSROW>
       <ATSROW num="2" replicateID="655362" database="bank" owner="testadm" table="customer"
              operation="Insert">
         <REPLICATED>
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                          cdrTimeString="2008-04-11T14:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">262</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
           </DATA>
         </REPLICATED>
       </ATSROW>
     </ATSROWS>
   </ATS>
</ERFILE>
```

The following example shows the corresponding RIS file for the failed transaction shown in the ATS example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <RIS version="1">
     <SOURCE id="20" name="g_cdr_ol_2" commitTime="2008-04-11T14:08:57"/>
     <TARGET id="30" name="g_cdr_ol_3" receiveTime="2008-04-11T14:08:57"/>
     <RISROWS>
       <RISROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
              operation="Insert">
         <CDRERROR num="0"/>
         <SQLERROR num="-668"/>
         <ISAMERROR num="-1"/>
         <SPLCODE num="63"/>
         <LOCAL>
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1206852121"
                          cdrTimeString="2008-04-11T12:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">261</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
           </DATA>
         </LOCAL>
         <REPLICATED>
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                          cdrTimeString="2008-04-11T14:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">261</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
          </DATA>
        </REPLICATED>
       </RISROW>
     </RISROWS>
     <TXNABORTED ATSFile="/tmp/ats.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.4.xml"
                generateATSFile="true"/>
  </RIS>
</ERFILE>
```

XML Tags

XML tags are used in ATS and RIS files that are generated in XML format.

Table 19. XML tags in ATS and RIS files

Tag name	Description	Attributes	Parent tag	Child tags
ERFILE	Top level tag for ATS and RIS files	`version`: XML file format version number.	None	ATS RIS
ATS	Parent tag for ATS files	`version`: ATS file format version number.	ERFILE	TRANSACTION ATSROWS
RIS	Parent tag for RIS files	• `version`: RIS file format version number. • `fromSource`: Set to `true` if the RIS file is generated at the source server.	ERFILE	SOURCE TARGET RISROWS TXNABORTED TXNCOMMITTED
TRANSACTION	Contains the name of the RIS file (if it exists) and the number of rows processed before the transaction was aborted.	• `RISFile`: The name of the RIS file, if it was created. • `generateRISFile`: Set to `true` if an RIS file exists for this aborted transaction.	ATS	SOURCE TARGET MESSAGE CDRERROR SQLERROR

Table 19. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
		- `processedRows`: Number of rows processed before the transaction was aborted.		ISAMERROR SPLCODE
ATSROWS	Contains the replicated aborted rows	None	ATS	ATSROW
SOURCE	Contains source server information	- `id`: Server ID. - `name`: Server group name. - `commitTime`: Transaction commit time.	TRANSACTION RIS	None
TARGET	Contains target server information	- `id`: Server ID. - `name`: Server group name. - `receiveTime`: Transaction receive time.	TRANSACTION RIS	None
SQLERROR	Contains the SQL error code	`num`: Error number.	TRANSACTION RISROW	None
ISAMERROR	Contains the ISAM error code	`num`: Error number.	TRANSACTION RISROW	None
CDRERROR	Contains the data sync error code	- `num`: Error number. - `description`: Error description.	TRANSACTION RISROW	None
MESSAGE	Contains the notification message	None	TRANSACTION RISROW	None
SPLCODE	Contains the SPL code number if a stored procedure conflict rule is being used	`num`: SPL code number.	TRANSACTION RISROW	None

Table 19. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
RISROWS	Contains the local and replicated aborted rows	None	RIS	RISROW
RISROW	Contains information about local or replicated row data for one aborted row	• `num`: Row sequence number. • `replicateID`: Replicate ID. • `database`: Database name. • `owner`: Table owner name. • `table`: Table name. • `operation`: DML operation type.	RISROWS	MESSAGE CDRERROR SQLERROR ISAMERROR SPLCODE MESSAGE LOCAL REPLICATED
LOCAL	Contains the local row data for an aborted row	None	RISROW	SHADOWCOLUMNS DATA
REPLICATED	Contains replicated row data for an aborted row	None	ATSROW RISROW	SHADOWCOLUMNS DATA
ATSROW	Contains information for one replicated aborted row	• `num`: Row sequence number. • `replicateID`: Replicate ID. • `database`: Database name. • `owner`: Table owner name. • `table`: Table name. • `operation`: DML operation type.	ATSROWS	REPLICATED
SHADOWCOLUMNS	Optional shadow column values for local and replicated rows	• `serverID`: Server ID. • `serverName`: Server group name.	LOCAL REPLICATED	None

Table 19. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
		- `cdrTimeInt`: The **cdrtime** column value in integer format (GMT time). - `cdrTimeString`: Time in string format. For example: 2008-11-08T20:16:25.		
DATA	Contains aborted row data	`dataExists`: Identifies whether data exists for this row or not.	ATSROW RISROW	COLUMN
COLUMN	Contains column data for an aborted row	- `name`: The column name. - `dataExists`: Identifies whether data is displayed for this column or not. - `isLOBorUDT`: Set to `true` if the column is of type UDT, smart large object or simple large object. If set to `true`, data for the column is skipped and the `dataExists` value is set to `false`. - `isHex`: Set to `true` if column data is displayed in hex format because Enterprise Replication does not have enough information to interpret the row data. - `isNull`: Set to `true` if the column value is NULL. Set to `false`	DATA	None

Table 19. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
		if the column has a valid value or an empty string.		
TXNABORTED	Indicates that the replicated transaction was aborted	• `ATSFile`: The name of the ATS file if the transaction was aborted and an ATS file was created for this aborted row. • `generateATSFile`: Set to `true` if an ATS file was created. • `TxnErr`: Error description for the aborted transaction.	RIS	None
TXNCOMMITTED	Indicates that the replicated transaction was committed	`totalRows`: Total number of rows processed.	RIS	None

ATS and RIS Text File Contents

The information about failed replicated transactions that are shown in ATS and RIS text files is listed in rows that are prefaced by information labels.

The first three characters in each line of the ATS and RIS file describe the type of information for the line, as the following table defines. The first four labels apply to both ATS and RIS files. The last three labels apply to only RIS files.

Table 20. Information labels

Label	Name	Description
TXH	Transaction heading	This line contains information from the transaction header, including the sending server ID and the commit time, the receiving server ID and the received time, and any Enterprise Replication, SQL, or ISAM error information for the transaction.
RRH	Replicated row heading	This line contains header information from the replicated rows, including the row number within the transaction, the group ID, the replicate ID (same as replicate group ID if replicate is not part of any replicate group), the database, owner, table name, and the database operation.

Table 20. Information labels (continued)

Label	Name	Description
RRS	Replicated row shadow columns	This line contains shadow column information from replicated rows, including the source server ID and the time when the row was updated on the source server. This line is printed only if the replicate is defined with a conflict-resolution rule.
RRD	Replicated row data	This line contains the list of replicated columns in the same order as in the SELECT statement in the cdr define replicate command. Each column is separated by a pipe character (|) and displayed in ASCII format. When the spooling program encounters severe errors (for example: cannot retrieve the replicate ID for the replicated row; unable to determine the replicated column type, size, or length), it displays this row data in hexadecimal format. The spooling program also displays the row data in hexadecimal format if a row includes replicated UDT columns.
LRH	Local-row header	RIS only. Indicates if the local row is found in the delete table and not in the target table
LRS	Local-row shadow columns	RIS only. Contains the server ID and the time when the row was updated on the target server This line is printed only if the replicate is defined with a conflict resolution rule.
LRD	Local-row data	RIS only. Contains the list of replicated columns extracted from the local row and displayed in the same order as the replicated row data. Similar to the replicated row data, each column is separated by a '|' and written in ASCII format. When the spooling program encounters severe errors (for example: cannot retrieve the replicate ID for the replicated row; unable to determine the replicated column type, size, or length) or the table includes UDT columns (whether defined for replication or not), it displays the replicated row data in hexadecimal format. In this case, the local row data is not spooled.

Changed Column Information

If you define a replicate to only replicate columns that changed, the RRD entry in the ATS and RIS file shows a `?` for the value of any columns that are not available. For example:

```
RRD 427|amsterdam|?|?|?|?|?|?|?|?|?|?|?|?
```

For more information, see Replicate only changed columns on page 106.

BLOB and CLOB Information

If a replicate includes one or more BLOB or CLOB columns, the RRD entry in the ATS and RIS file displays the smart large object metadata (the in-row descriptor of the data), not the smart large object itself, in hexadecimal format.

BYTE and TEXT Information

When the information recorded in the ATS or RIS file includes BYTE or TEXT data, the replicated row data (RRD) information is reported, as the following examples show.

Example 1

```
<1200, TEXT, PB 877(necromsv) 840338515(00/08/17 20:21:55)>
```

In this example:

- `1200` is the size of the data.
- `TEXT` is the data type (it is either BYTE or TEXT).
- `PB` is the storage type (`PB` when the BYTE or TEXT is stored in the tblspace, `BB` for blobspace storage).
- The next two fields are the server identifier and the time stamp for the column if the conflict-resolution rule is defined for this replicate and the column is stored in a tblspace.

```
<500 (NoChange), TEXT, PB 877(necromsv) 840338478(00/08/17 20:21:18)>
```

Example 2

In this example, `500 (NoChange)` indicates that the TEXT data has a size of 500, but the data is not changed on the source server. Therefore, the data is not sent from the source server.

Example 3

```
<(Keep local blob),75400, BYTE, PB 877(necromsv) 840338515(00/08/17 20:21:55)>"
)
```

In this example, (`Keep local blob`) indicates that the replicated data for this column is not applied on the target table, but instead the local BYTE data was kept. This usually happens when time stamp conflict resolution is defined and the local column has a time stamp greater than the replicated column.

UDT Information

If a replicate includes one or more UDT columns, the RRD entry in the ATS and RIS files displays the row data in delimited format as usual, except the string `<skipped>` is put in place of UDT column values. For example, the following row shows information about a table with columns of type INTEGER, UDT, CHAR(10), and UDT:

```
RRD 334|<skipped>|amsterdam|<skipped>
```

TimeSeries information

If a replicate includes a **TimeSeries** column, an RTS row displays the time series instance ID and the timestamp of the element. If the failed replicated transaction includes a TimeSeries routine that affects a range of elements, both the starting and ending timestamps are shown. The following example shows three failed replication transactions that include a **TimeSeries** column:

```
TXH Source ID:100 / Name:g_delhi / CommitTime:12-01-27 12:27:39
TXH Target ID:200 / Name:g_bombay / ReceiveTime:12-01-27 12:27:39
----------
```

```
RRH Row:1 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSInsert
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Timestamp=12-01-27 12:27:39

RRH Row:2 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSDelete
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Timestamp=12-01-27 12:27:39

RRH Row:2 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSDelRange
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Begin Timestamp=12-01-27 12:27:39, End Timestamp=12-01-27 12:27:39
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Disabling ATS and RIS File Generation

You can prevent the generation of ATS or RIS files, or both.

About this task

To prevent the generation of both ATS and RIS files, set the CDR_DISABLE_SPOOL environment variable to `1`.

To prevent the generation of either ATS or RIS files, set the ATS or RIS directory to **/dev/null** (UNIX™) or **NUL** (Windows™) with the **cdr define server** or **cdr modify server** commands.

Related reference

cdr modify server on page 421

cdr define server on page 357

CDR_DISABLE_SPOOL Environment Variable on page 521

Suppressing Data Sync Errors and Warnings

You prevent certain data sync errors and warnings from appearing in ATS and RIS files by using the CDR_SUPPRESS_ATSRISWARN configuration parameter.

For more information on the CDR_SUPPRESS_ATSRISWARN configuration parameter, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter on page 515.

For a list of error and warning messages that you can suppress, see Data sync warning and error messages on page 610.

Preventing Memory Queues from Overflowing

In a well-tuned Enterprise Replication system, the send queue and receive queue do not regularly overflow from memory to disk. However, if the queues in memory fill, the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of transaction records, replicate information, and row data. Spooled transaction records and replicate information are stored in the transaction tables and the replicate information tables in a single dbspace. Spooled row data is stored in one or more sbspaces.

About this task

The following situations can cause Enterprise Replication to spool to disk:

- Receiving server is down or suspended.
- Network connection is down.

  If the receiving server or network connection is down or suspended, Enterprise Replication might spool transaction buffers to disk.

  To check for a down server or network connection, run cdr list server on a root server. This command shows all servers and their connection status and state.
- Replicate is suspended.

  If a replicate is suspended, Enterprise Replication might spool transaction buffers to disk.

  To check for a suspended replicate, run cdr list replicate. This command shows all replicates and their state.
- Enterprise Replication is replicating large transactions.

  Enterprise Replication is optimized to handle small transactions efficiently. Very large transactions or batch jobs force Enterprise Replication into an exceptional processing path that results in spooling. For best results, avoid replicating these types of transactions.
- Logical log files are too small or too few.

  If the logical log files are too small or the number of logical log files is too few, Enterprise Replication is more likely to spool transaction buffers to disk.
- Server is overloaded.

  If a server is low on resources, Enterprise Replication might not be able to hold all transactions that are replicating from a source server in memory during processing, and the transactions spool to disk.

  If transactions spool to disk, check the system resources; in particular, check disk speed, RAM, and CPU resources.

Related reference

cdr list server on page 402

cdr list replicate on page 395

Related information

Send queues and receive queues on page 14

Setting Up Send and Receive Queue Spool Areas on page 66

Transaction processing impact on page 20

Logical Log Configuration Guidelines on page 64

Handle potential log wrapping

The potential for log wrap occurs when Enterprise Replication log processing lags behind the current log and the Enterprise Replication replay position is in danger of being overrun.

There are two log positions you should be aware of: the snoopy log position, which is the log position that keeps track of transactions being captured for replication, and the log replay position, which is the log position that keeps track of which transactions have been applied.

A potential log wrap situation is usually caused by the logical logs being misconfigured for the current transaction activity or by the Enterprise Replication system having to spool more than usual. More-than-usual spooling could be caused by one of the following situations:

- A one-time job might be larger than normal and thus require more log space.
- One of the target servers is currently unavailable and more spooling of replicated transactions is required.
- The spool file or paging space could be full and needs to be expanded.

You can configure how Enterprise Replication responds to a potential log wrap situation by specifying one or more of the following solutions, in order of priority, with the CDR_LOG_LAG_ACTION configuration parameter:

- Block user transactions until Enterprise Replication log processing advances far enough that the danger of log wrapping is diminished. Blocking user transactions prevents the current log position from advancing. This solution increases user response time. When user transactions are blocked, event alarm 30 unique ID 30002 is raised and the following message appears in the online log:

  ```
  DDR Log Snooping - DDRBLOCK phase started, userthreads blocked
  ```
- Compress the logical logs and save them to a log staging directory. Log files in the staging directory are deleted after they are no longer required by Enterprise Replication. You must specify the location and maximum size of the log staging directory. This solution uses very little additional disk space to temporarily save log files until the danger of log wrapping is over. The staged log files are deleted after advancing the log replay position.

  If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.
- Dynamically add logical logs. This solution requires enough free space to be available in the logical log dbspace to add dynamic logs. You can specify how many dynamic logical logs to add. You must manually drop the dynamic log files when the danger for log wrapping is over.
- Ignore the potential for log wrap. This solution shuts down Enterprise Replication when an overrun of the snoopy log replay position is detected. Enterprise Replication continues to function if the log replay position is overrun. If the snoopy replay position is overrun, Enterprise Replication is stopped, event alarm 47 is raised, and the following message appears in the message log file:

  ```
  WARNING:  The replay position was overrun, data may not be replicated.
  ```

If the replay position is overrun, restart Enterprise Replication with the cdr cleanstart command to reset replay position to current log position and synchronize the data.

- Stop Enterprise Replication on the affected server as soon as it is detected that the log replay position is running behind. When you are ready to restart Enterprise Replication it is necessary to run the cdr cleanstart command only if the log replay position was overrun.

For example, you can specify that during a potential log wrap situation, Enterprise Replication stages compressed logical logs. If the log staging directory reaches its maximum size, then logical logs are added. If the maximum number of logical logs are added, then Enterprise Replication blocks user transactions. Not all options can be combined together in every possible priority order. For example, specifying to stop Enterprise Replication, to ignore the potential for log wrap, or to block user actions must always be either the only option or the last option in the list.

Related reference

CDR_LOG_LAG_ACTION configuration parameter on page 504

CDR_LOG_STAGING_MAXSIZE Configuration Parameter on page 507

CDR_MAX_DYNAMIC_LOGS Configuration Parameter on page 509

Related information

Logical Log Configuration Guidelines on page 64

Monitoring Disk Usage for Send and Receive Queue Spool

Periodically monitor disk usage for the dbspace.

About this task

The sbspace that Enterprise Replication uses to spool the queues to disk is specified by the CDR_QDATA_SBSPACE configuration parameter.

To check disk usage for the spooling sbspace, run one or more of the following commands:

- onstat -g rqm SBSPACES
- onstat -d

> **Tip:** When you use the onstat -d command to monitor disk usage, the S flag in the **Flags** column indicates an sbspace. For each sbspace chunk, the first row displays information about the whole sbspace and user-data area. The second row displays information about the metadata area.

- The oncheck command with the -cs,-cS, -ce, -pe, -ps, and -pS options

Related reference

onstat -g rqm: Prints statistics for RQM queues on page 567

CDR_QDATA_SBSPACE Configuration Parameter on page 512

Related information

Manage sbspaces on page

onstat -d command: Print chunk information on page

The oncheck Utility on page

Increasing the Sizes or Numbers of Storage Spaces

About this task

If you notice that the Enterprise Replication dbspace or sbspace is running out of disk space, you can increase the size of the space by adding chunks to the space. You can also add additional sbspaces for Enterprise Replication.

To add a chunk to a dbspace, use **onspaces -a**. For example, to add a 110 kilobyte chunk with an offset of 0 to the **er_dbspace** dbspace, enter:

```
onspaces -a er_dbspace -p /dev/raw_dev2 -o 0 -s 110
```

To add a chunk to an sbspace, use the same **onspaces** command above, however you can specify more information about the chunk that you are adding. After you add a chunk to the sbspace, you must perform a level-0 backup of the root dbspace and the sbspace.

See the sections on adding chunks to dbspaces and sbspaces in the *HCL® Informix® Administrator's Guide* and the *HCL® Informix® Administrator's Reference* for more information.

To increase the number of sbspaces that can be used for Enterprise Replication, create new sbspaces with the **onspaces -c -S** command and then add their names to the CDR_QDATA_SBSPACE configuration parameter with the **cdr add onconfig** command. For more information, see cdr add onconfig on page 289.

Recovering when Storage Spaces Fill

When the Enterprise Replication dbspace runs out of disk space, Enterprise Replication raises an alarm and writes a message to the log. When the sbspace runs out of disk space, Enterprise Replication hangs. In either case, you must resolve the problem that is causing Enterprise Replication to spool (Preventing Memory Queues from Overflowing on page 212) or you must allocate additional disk space (Increasing the Sizes or Numbers of Storage Spaces on page 216) before you can continue replication.

Common configuration problems

If you experience problems setting up Enterprise Replication, check the configuration of your environment and database.

To solve configuration problems:

- Make sure that you created an sbspace for the row data and set the CDR_QDATA_SBSPACE in the `onconfig` file.

  For more information, see Setting Up Send and Receive Queue Spool Areas on page 66 and CDR_QDATA_SBSPACE Configuration Parameter on page 512.
- Verify that the trusted environment is set up correctly.

  For more information, see Configuring secure ports for connections between replication servers on page 59.
- Verify that your `sqlhosts` file is set up properly on each server that participates in replication. You must set up database server groups in the `sqlhosts` file.

  For more information, see Creating sqlhost group entries for replication servers on page 58.
- Verify the format of the `sqlhosts` file.

  The network connection (not the shared memory connection) entry must be immediately after the database server group definition. If the network connection entry is not immediately after the database server group definition, you might see the following error when you run cdr define server:

  ```
  command failed -- unable to connect to server specified (5)
  ```

  You might also see a message like the following in the message log for the target server:

  ```
  Reason: ASF connect error (-25592)
  ```
- Make sure that the unique identifier for each database server (i= in the **options** field of the `sqlhosts` information) is consistent across all nodes in the domain.

  For more information, see Creating sqlhost group entries for replication servers on page 58.
- Verify that the operating system times of the database servers that participate in the replicate are synchronized.

  For more information, see Time synchronization on page 74.
- Make sure that the database server has adequate logical log disk space. If the database server does not have enough logical log space at initialization, you see the following error:

  ```
  command failed -- fatal server error (100)
  ```
- Check the files in the `$INFORMIXDIR` directory to see if a problem occurred when the database server built the SMI tables.
- Make sure that the databases on all database server instances that are involved in replication are set to logging (unbuffered logging is recommended).

  For more information, see Unbuffered Logging on page 23.
- For replicates that use any conflict-resolution rule except ignore and always-apply, make sure that you define shadow columns (CRCOLS) for each table that is involved in replication.

  For more information, see Preparing Tables for Conflict Resolution on page 77.
- If you defined a participant using SELECT * from *table_name* statement, make sure that the tables are identical on all database servers that are defined for the replicate.

For more information, see Participant definitions on page 100 and Participant and participant modifier on page 257.

- Verify that each replicated column in a table on the source database server has the same data type as the corresponding column on the target server.

  Enterprise Replication does not support replicating a column with one data type to a column on another database server with a different data type.

  The exception to this rule is cross-replication between simple large objects and smart large objects.

  For more information, see Replication and data types on page 29.

- Verify that all tables defined in a replicate have a replication key.

  For more information, see Unique key for replication on page 25.

- If high-availability clusters are also in use in the domain, then all row data sbspaces must be created with logging by using the -Df "LOGGING=ON" option of the onspaces command.

  For more information, see Row Data sbspaces on page 67 and the *HCL® Informix® Administrator's Guide*.

Troubleshooting Tips for Alter Operations

Alter operations on replicated tables might result in errors.

The following problems illustrate common issues with performing alter operations on replicated tables:

- **Problem:** You receive an error that the replicate is not defined after running the following command:

  ```
  cdr alter -o test:tab
  Error:Replicate(s) not defined on table test:.tab
  ```

  The owner name is missing from the table name, **test:tab**.

  Solution: Include the table owner name, for example:

  ```
  cdr alter -o test:user1.tab
  ```

- **Problem:** You receive an error that the replicated table is in alter mode after running the following command:

  ```
  > insert into tab values(1,1);

  19992: Cannot perform insert/delete/update operations on a replicated table
  while the table is in alter mode
  Error in line 1 Near character position 27
  >
  ```

  The table (**tab**) is in alter mode. DML operations cannot be performed while the table is in alter mode.

  Solution: Wait for the table to be altered and then issue the DML operation. If no alter statement is in progress against the table, then unset alter mode on the table using the **cdr alter --off** command. For example:

  ```
  cdr alter --off test:user1.tab
  ```

You can check the alter mode status using the **oncheck -pt** command. For example:

```
$ oncheck -pt db1:user1.t1

TBLspace Report for db1:user1.t1

    Physical Address             1:63392
    Creation date                02/01/2011 16:02:00
    TBLspace Flags               400809     Page Locking
                                            TBLspace flagged for replication
                                            TBLspace flagged for CDR alter mode
                                            TBLspace use 4 bit bit-maps
    Maximum row size               4
...
```

- **Problem:** How can you tell if a replicate is a mastered replicate?

  Solution: You can check the alter mode status using the **oncheck -pt** command. For example:

  ```
  oncheck -pt test:nagaraju.tab
  ```

- **Problem:** How can you tell if a replicate is a mastered replicate?

  Solution: When you execute the **cdr list repl** command, it shows that the REPLTYPE is Master for master replicates. For example:

  ```
  $cdr list repl
  CURRENTLY DEFINED REPLICATES
  -------------------------------
  REPLICATE: rep2
  STATE: Active ON:delhi
  CONFLICT: Timestamp
  FREQUENCY: immediate
  QUEUE SIZE: 0
  PARTICIPANT: test:nagaraju.tab12
  OPTIONS: transaction,ris,ats,fullrow
  REPLTYPE: Master

  REPLICATE: rep1
  STATE: Active ON:delhi
  CONFLICT: Timestamp
  FREQUENCY: immediate
  QUEUE SIZE: 0
  PARTICIPANT: test:nagaraju.tab
  OPTIONS: transaction,ris,ats,fullrow
  ```

  In the above output, **rep1** is defined as a non-master replicate and **rep2** is defined as master replicate.

- **Problem:** An alter operation on a replicated table fails.

  For example:

  ```
  $dbaccess test -

  Database selected.

  > alter table tab add col4 int;

  ```

```
19995: Enterprise Replication error encountered while setting alter mode. See
message log file to get the Enterprise Replication error code
Error in line 1Near character position 27
>
```

The message log output is:

```
12:36:09 CDRGC: Classic replicate rep1 found on the table test:nagaraju.tab
12:36:09 CDRGC:Set alter mode for replicate rep1
12:36:09 GC operation alter mode set operation on a replicated table failed:
Classic replicate(s) (no mastered dictionary) found on the table.
```

Solution: The above message shows that there is a classic replicate, **rep1**, defined on the table (**tab**). Adding a new column to a replicated table is allowed when only master replicates are defined for the table.

To perform the above alter operation, first convert the classic replicate to a master replicate. You can convert the replicate definition of **rep1** to a master replicate by issuing the following command:

```
cdr remaster -M g_delhi rep1 "select * from tab"
```

Now look at the **cdr list repl** output:

```
$cdr list repl
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE: rep1
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: rep2
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab12
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: Shadow_4_rep1_GMT1112381058_GID100_PID29935
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

You can see that **repl1** has been converted to a master replicate. You can also see that a new replicate definition, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, was also created against the table (**tab1**). Notice the last two fields of the output for **Shadow_4_rep1_GMT1112381058_GID100_PID29935**:

```
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

The Shadow attribute indicates that this replicate is a shadow replicate, and PARENT REPLICATE: **rep1** shows that this is a shadow replicate for the primary replicate **rep1**. Notice that the Master attribute is not present for this replicate definition. This shadow replicate is actually the old non-master replicate. The **cdr remaster** command created a new master replicate, **rep1**, for the table **tab** and converted the old non-master replicate (**rep1**) to a shadow replicate for the new master replicate.

This table is not yet ready to be altered because there is still a non-master replicate, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, defined for the table, **tab**. You must wait for **Shadow_4_rep1_GMT1112381058_GID100_PID29935** to be deleted automatically by Enterprise Replication after all the data queued for this shadow replicate is applied at all the replicate participants. This process can take some time. Alternatively, if you are sure that there is no data pending for this old non-master replicate, then you can issue the **cdr delete repl** command against **Shadow_4_rep1_GMT1112381058_GID100_PID29935**.

After making sure that **Shadow_4_rep1_GMT1112381058_GID100_PID29935** no longer exists, you can attempt the **ALTER TABLE tab add col4 int;** statement against the table.

Enterprise Replication Event Alarms

Certain Enterprise Replication errors and other actions generate event alarms. You can use event alarms specific to Enterprise Replication to automate many administrative tasks.

You can set your alarm program script to capture Enterprise Replication class IDs and messages and initiate corrective actions or notifications for each event. For example, you can add a chunk to the queue data sbspace or dbspace if you detect (using class ID 31) that the storage space is full.

Most event alarms operate in the background. For events that operate in the foreground, the session that triggered the alarm is suspended until the alarm program execution completes. For information on setting alarm program scripts to capture events, see Event Alarms on page .

Many Enterprise Replication event alarms are enabled by default, but most state change event alarms are disabled by default. You can control which Enterprise Replication event alarms are enabled with the CDR_ALARMS environments variable.

The following table lists the information about Enterprise Replication event alarms:

- The class ID is an integer value identifying the category of the event.
- The event ID is a unique identifier for the specific message.
- The class message provides general information about the event.
- The specific message provides detailed information about the event.
- The severity describes the seriousness of the event on a scale from 1 to 5, where 5 is the most serious.
- Whether the event operates in the foreground and explanations for the events.
- Whether the event is disabled by default.

Table 21. Enterprise Replication Event Alarms

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 Event ID: 30002	Class message: DDR subsystem notification Specific message: DDR Log Snooping - Catchup phase started, userthreads blocked	3	User transactions are being blocked to prevent the database server from overwriting a logical log that Enterprise Replication has not yet processed. **Online log:** The following message appears in the online log: DDR Log Snooping - DDRBLOCK phase started, userthreads blocked **ER state:** Active and replicating data. User transactions are temporarily blocked. **User action:** None. For information about preventing this situation, see Handle potential log wrapping on page 214.
Class ID: 30 Event ID: 30003	Class message: DDR subsystem notification Specific message: DDR Log Snooping - Catchup phase completed, userthreads unblocked	3	User transactions are no longer blocked. **Online log:** The specific message also appears in the online log. **ER state:** Active and replicating data. **User action:** None.
Class ID: 30 Event ID: 30004	Class message: DDR subsystem failure Specific message: WARNING: The replay position was overrun, data may not be replicated.	4	The log replay position was overwritten. **Online log:** The following message appears in the online log: WARNING: The replay position was overrun, data may not be replicated. **ER state:** Active and replicating data. Enterprise Replication shuts down if the log read position also gets overwritten. If Enterprise Replication shuts down, event alarm 47 is raised.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
			User action: For information about preventing this situation, see Handle potential log wrapping on page 214.
Class ID: 30 Event ID: 30005	Class message: DDR Subsystem notification Specific message: CDR DDR: Log staging disk space usage reached its allowed configured maximum size *size* (KB). Temporarily disabling log staging.	3	The disk space where logs are stored reached its maximum size. **Online log:** The following message appears in the online log: CDR DDR: Log staging disk space usage reached its allowed configured maximum size *size* (KB). Temporarily disabling log staging. **ER state:** Active and running. Enterprise Replication uses the next configured logical log lag action to protect the replay position. If no other log lag action is configured, the replay position can be overrun. If Enterprise Replication shuts down due to replay position being overrun, restart Enterprise Replication using cdr cleanstart command and resynchronize the data. **User action:** Consider increasing the maximum disk space configured for log staging using the CDR_LOG_STAGING_MAXSIZE configuration parameter. The value for the CDR_LOG_STAGING_MAXSIZE configuration parameter can be updated while the server is active using the following command: `onmode -wf CDR_LOG_STAGING_MAXSIZE=`*size*
Class ID: 30 Event ID: 30006	Class message: DDR Subsystem notification Specific message: CDR: Created staging file *filename* for log unique id *unique_log_id*	3	The log staging file was created. **Online log:** The following message appears in the online log: CDR: Created staging file *filename* for log unique id *uniqui_log_id* **ER state:** Enterprise Replication is active and staging log files because a log lag state was detected.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
			User action: If high-availability secondary servers are configured, consider copying log files to the secondary server. See Transferring log files to a high-availability cluster secondary server when using ER on page 507.
Class ID: 30 Event ID: 30007	Class message: DDR Subsystem notification Specific message: CDR: Completed processing log unique id *unique_log_id*. Deleted log staging file *filename*	2	A log staging file was deleted. **Online log:** The following message appears in the online log: CDR: Completed processing log unique id *unique_log_id*. Deleted log staging file *filename* **ER state:** Active and replicating data. **User action:** If staged log files are being copied to high-availability secondary servers, consider deleting the log staged log file name specified in the alarm message and the related token log file. See Transferring log files to a high-availability cluster secondary server when using ER on page 507.
Class ID: 30 Event ID: 30008	Class message: DDR Subsystem notification Specific message: CDR: Deleted all staging files from log staging directory.	2	The staging files were deleted from the log staging directory. **Online log:** The following message appears in the online log: CDR: Deleted all staging files from log staging directory. **ER state:** Active or deleted. Enterprise Replication deletes all files in the log staging directory when they are no longer required. The log files are deleted when any of the following occur: • Enterprise Replication is deleted on the local server. • After the cdr cleanstart command is run. • When the value of the LOG_STAGING_DIR configuration parameter is changed (any log

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
			files that exist in the previous directory are also deleted). • When Enterprise Replication is defined. **User action:** If staged log files are being manually copied to high-availability secondary server then delete all staged log files on the secondary servers. See Transferring log files to a high-availability cluster secondary server when using ER on page 507.
Class ID: 31 Event ID: 31001	Class message: ER stable storage pager sbspace is full Specific message: CDR Pager: Paging File full: Waiting for additional space in *sbspace_name*	4	This event runs in the foreground. The grouper paging sbspace ran out of space. **ER state:** Active and waiting for the space to be added to the sbspace name specified in alarm-specific message. **User action:** Add a chunk to the specified sbspace. For information about preventing this situation, see Increasing the Sizes or Numbers of Storage Spaces on page 216.
Class ID: 31 Event ID: 31002	Class message: ER stable storage queue sbspace is full Specific message: CDR QUEUER: Send Queue space is FULL - waiting for space in *sbspace_name*. CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QDATA_SBSPACE	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the sbspace listed. **User action:** Add a chunk to the specified sbspace. If the message specifies CDR_QDATA_SBSPACE, add a chunk to one or more of the sbspaces specified by the CDR_QDATA_SBSPACE configuration parameter. For information about preventing this situation, see Recovering when Storage Spaces Fill on page 216.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 31 Event ID: 31003	Class message: ER stable storage queue dbspace is full Specific message: CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QHDR_DBSPACE..	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the dbspace specified by the CDR_DBSPACE configuration parameter. **User action:** Add a chunk to the dbspace specified by the CDR_DBSPACE configuration parameter. For information about preventing this situation, see Recovering when Storage Spaces Fill on page 216.
Class ID: 32 Event ID: 32002	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper Fanout thread is aborting.	4	The grouper fanout thread is quitting. **ER state:** Enterprise Replication was shut down internally. Event alarm 47 is also raised. **User action:** Restart Enterprise Replication using the cdr start command.
Class ID: 32 Event ID: 32003	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper Evaluator thread is aborting.	4	The grouper evaluator thread is quitting. **ER state:** Active and replicating transactions. **User action:** Stop Enterprise Replication with the cdr stop command and restart it using the cdr start command.
Class ID: 32 Event ID:	Class message: ER: error detected in grouper sub component Specific message:	4	The grouper subcomponent cannot copy the transaction into the send queue. **ER state:** Active and replicating transactions.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
32004	CDR: Could not copy transaction at log id *log_unique_id* position *log_position*. Skipped.		**User action:** Shut down Enterprise Replication by running the cdr stop command, clear the receive queue and restart replication by running the cdr cleanstart command, and then synchronize the data by running the cdr check replicateset command with the --repair option.
Class ID: 32 Event ID: 32005	Class message: ER: error detected in grouper sub component Specific message: CDR: Paging error detected.	4	The grouper subcomponent detected a paging error. **ER state:** Inactive. **User action:** Restart Enterprise Replication by running the cdr start command.
Class ID: 32 Event ID: 32006	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper: Local participant (*participant_name*) stopped for the replicate *replicate_name* (or exclusive replicate set), table (*database*:*owner*.*table*). Data may be out of sync. If replicated column definition was modified then please perform the alter operation at all the replicate participants, remaster the replicate definition then restart the replicate (or exclusive replicate set) definition for the local participant with the data sync option (-S).	4	If the grouper subcomponent is not able to convert the replicated row data from the local dictionary format to the master dictionary format, the grouper stops the local participant from the corresponding replicate (or exclusive replicate set) definition and invokes this event alarm.
Class ID: 32 Event ID:	Class message: ER: error detected in grouper sub component	3	The grouper subcomponent did not roll back a transaction to a savepoint. **ER state:** Active and replicating transactions.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
32007	Specific message: CDR *CDR_subcomponent_name*: Could not apply undo properly. SKIPPING TRANSACTION. TX Begin Time: *datetime* TX Restart Log Id: *log_id* TX Restart Log Position: *log_position* TX Commit Time: *datetime* TX End Log Id: *log_id* TX End Log Position: *log_position*		**User action:** Run the cdr check replicateset command with the --repair option to make sure that the data is consistent.
Class ID: 33 Event ID: 33001	Class message: ER: error detected in data sync sub component Specific message: Received aborted transaction, no data to spool.	2	Data sync received a transaction that was aborted in the first buffer, so the transaction cannot be spooled to an ATS or RIS file. **ER state:** Active and replicating transactions. **User action:** Run the cdr check replicateset command with the --repair option to make sure that the data is consistent.
Class ID: 33 Event ID: 33002	Class message: ER: error detected in data sync sub component Specific message: CDR DS *thread_name* thread is aborting.	4	The data sync thread is quitting. **ER state:** Active and replicating transactions. **User action:** Run the cdr check replicateset command with the --repair option to make sure that the data is consistent.
Class ID: 33 Event ID:	Class message: ER: error detected in data sync sub component Specific message:	3	A table in alter mode is blocking the application of transactions. While the table is in alter mode, Enterprise Replication cannot apply transactions that involve this table or are in a referential relationship with this table. Enterprise Replication also cannot apply subsequent

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
33002	Table in alter mode is blocking application of transactions. Table: *dbname*:'*owner*'.*tabname*.		transactions from the site where the failed transaction originated. **User action:** Run `cdr alter --off dbname:owner.tabname`.
Class ID: 34 Event ID: 34001	Class message: ER: error detected in queue management sub component Specific message: CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground. RQM cannot find the replicate in the global catalog for which it has a transaction. **ER state:** Active and replicating transactions. **User action:** Run the cdr check replicateset command with the --repair option to make sure that the data is consistent.
Class ID: 35 Event ID: 35001	Class message: ER: error detected in global catalog sub component Specific message: CDR GC peer request failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Execution of the control command requested by the peer server failed at the local server. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.
Class ID: 35 Event ID: 35002	Class message: ER: error detected in global catalog sub component Specific message: CDR GC peer processing failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Control command execution at the peer server failed. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.
Class ID: 35 Event ID:	Class message: ER: error detected in global catalog sub component	3	The delete table was not dropped while the replicate was being deleted from the local participant. **ER state:** Active and replicating transactions.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
35004	Specific message: CDR: Could not drop delete table. SQL code *sql_error_code*, ISAM code *isam_error_code*. Table '*database:table*'. Please drop the table manually.		**User action:** Manually drop the delete table.
Class ID: 36 Event ID: 36001	Class message: ER: enterprise replication network interface sub component notification Specific message: Enterprise Replication: Connection to *servergroupname* closed. Reason: connection request received from an unknown server.	3	Enterprise Replication received a reconnect connection request from an unknown server. **ER state:** Active. **User action:** Check the connection requester server definition in the local server. If the definition is not available on the local server, the remote server definition was probably deleted on the local server by running the cdr delete server command, but the cdr delete server command was not run on the remote server. In this case, run the cdr delete server command on the remote server and, if necessary, redefine the server.
Class ID: 37 Event ID: 37001	Class message: ER: error detected while recovering Enterprise Replication Specific message: CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground; Enterprise Replication is blocked until this issue is resolved. **ER state:** Active and replicating transactions. **User action:** If the replicate ID is still valid and exists in **syscdr** catalog tables, run the cdr check replicateset command with the --repair option to make sure that the data is consistent.
Class ID: 38 Event ID: 38001	Class message: ER: resource allocation problem detected Specific message:	2	The specified Enterprise Replication component did not allocate memory. **ER state:** Active. **User action:** Perform these actions:

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
	CDR *CDR_subcomponent_name* memory allocation failed (*reason*).		1. Correct the resource issue. 2. Stop replication by running the cdr stop command. 3. Restart replication by running the cdr start command. 4. Make sure that the data is consistent by running the cdr check replicateset command with the --repair option.
Class ID: 39 Event ID: 39001	Class message: Please notify HCL® Informix® Technical Support Specific message: Log corruption detected or read error occurred while snooping logs.	4	A logical log is corrupted and cannot be processed by the log capture component. Event alarm 47 is also raised in this situation. **Online log:** The following message appears in the online log: Log corruption detected while snooping logs, logid=*log_unique_id* logpos=*log_position*. **ER state:** Inactive. **User action:** Clear the receive queue and restart replication by running the cdr cleanstart command, and then synchronize the data by running the cdr check replicateset command with the --repair option.
Class ID: 39 Event ID: 39002	Class message: Please notify HCL® Informix® Technical Support Specific message: CDR: Unexpected log record type *record_type* for subsystem *subsystem* passed to DDR.	4	A log record of unexpected type was passed to the log capture component. **ER state:** Active and replicating transactions. **User action:** Contact IBM® Software Support.
Class ID: 47	Class message: CDR is shutting down due to internal error: failure	4	Data sync threads encountered a memory allocation error while replaying replicated transactions and replication is stopped.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Event ID: 47001	Specific message: CDR is shutting down due to internal error: Memory allocation failed		**Online Log:** When the memory allocation error is discovered, the following message appears in the online log: CDR DS processes is aborting. Signaling CDR system to shutdown as it is low on resources. When Enterprise Replication is shutting down and the event alarm is being raised, the following message appears in the online log: CDR is shutting down due to internal error: Memory allocation failed **ER State:** No replicated transactions are lost while replication is stopped. **User Action:** To resume replication, solve the memory issue and run the cdr start command or shut down and restart the database server. If the replay position was overrun while replication was stopped, event alarm 75 is raised.
Class ID: 47 Event ID: 47005	Class message: CDR is shutting down due to internal error: failure Specific message: CDR is shutting down due to an internal error.	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** Try restarting Enterprise Replication using the cdr start command. If replay position overrun is detected and the cdr start command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the cdr cleanstart command and synchronize the data.
Class ID: 47 Event ID:	Class message: CDR is shutting down due to internal error: log lag state Specific message:	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** If replay position overrun was detected then restart Enterprise Replication using cdr cleanstart

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
47006	CDR DDR: Shutting down ER to avoid a DDRBLOCK situation.		command and synchronize the data. If the replay position was not overrun then restart Enterprise Replication using cdr start command; there is no need to synchronize the data. If replay position overrun is detected and the cdr start command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the cdr cleanstart command and synchronize the data.
Class ID: 48 Event ID: 48001	Class message: ATS and/or RIS files spooled to disk. Specific message: *file name*|*file name*.	3	One or more failed transactions caused the generation of one or more ATS or RIS files. The generated file names are listed in the specific message, separated with a pipe (|) character. **ER State:** Replication is continuing normally. **User Action:** To process the failed transactions, run the cdr repair command for each file, or run the cdr check replicateset command with the --repair option.
Class ID: 49 Event ID: 49001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is started on server *server_name*.	3	This event alarm is disabled by default. The cdr start command was run.
Class ID: 50 Event ID: 50001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is stopped on server *server_name*.	3	The cdr stop command was run.
Class ID:	Class message:	3	This event alarm is disabled by default.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
51 Event ID: 51001	A replication state change event has happened. Specific message: Enterprise Replication is suspended on server *server_name* .		The cdr suspend server command was run.
Class ID: 52 Event ID: 52001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is resumed on server *server_name*.	3	This event alarm is disabled by default. The cdr resume server command was run.
Class ID: 53 Event ID: 53001	Class message: A replication state change event has happened. Specific message: Server *server_name* is connected.	3	This event alarm is disabled by default. The cdr connect server command was run.
Class ID: 54 Event ID: 54001	Class message: A replication state change event has happened. Specific message: Server *server_name* is disconnected.	3	This event alarm is disabled by default. The cdr disconnect server command was run.
Class ID: 55 Event ID:	Class message: A replication state change event has happened. Specific message:	3	This event alarm is disabled by default. The cdr suspend replicate command was run.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
55001	Replication is suspended on replicate *replicate_name* on server *server_name* .		
Class ID: 56 Event ID: 56001	Class message: A replication state change event has happened. Specific message: Replication is suspended on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr suspend replicateset command was run.
Class ID: 57 Event ID: 57001	Class message: A replication state change event has happened. Specific message: Replication is resumed on replicate *replicate_name* on server *server_name* .	3	This event alarm is disabled by default. The cdr resume replicate command was run.
Class ID: 58 Event ID: 58001	Class message: A replication state change event has happened. Specific message: Replication is resumed on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr resume replicateset command was run.
Class ID: 59 Event ID:	Class message: A replication state change event has happened. Specific message:	3	This event alarm is disabled by default. The cdr start replicate command was run.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
59001	Replication is started on replicate *replicate_name* on server *server_name* .		
Class ID: 60 Event ID: 60001	Class message: A replication state change event has happened. Specific message: Replication is started on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr start replicateset command was run.
Class ID: 61 Event ID: 61001	Class message: A replication state change event has happened. Specific message: Replication is stopped on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr stop replicate command was run.
Class ID: 62 Event ID: 62001	Class message: A replication state change event has happened. Specific message: Replication is stopped on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr stop replicateset command was run.
Class ID: 63 Event ID:	Class message: A replication state change event has happened. Specific message:	3	This event alarm is disabled by default. The cdr modify replicate command was run.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
63001	Replication attribute is modified on replicate *replicate_name* on server *server_name* .		
Class ID: 64 Event ID: 64001	Class message: A replication state change event has happened. Specific message: Replication attribute is modified on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The cdr modify replicateset command was run.
Class ID: 65 Event ID: 65001	Class message: A replication state change event has happened. Specific message: Change in replicate *replicate_name* on server *server_name*: operation *action*, node[s] *participant_name*.	3	This event alarm is disabled by default. The cdr change replicate command was run to add or delete one or more participants.
Class ID: 66 Event ID: 66001	Class message: A replication state change event has happened. Specific message: Change in replicateset *replicateset_name* on server *server_name*: operation *action*, member[s] *replicate_name*.	3	This event alarm is disabled by default. The cdr change replicateset command was run to add or delete one or more replicates.
Class ID: 67	Class message:	3	This event alarm is disabled by default.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Event ID: 67001	A replication state change event has happened. Specific message: Server *server_name* is deleted.		The cdr delete server command was run.
Class ID: 68 Event ID: 68001	Class message: A replication state change event has happened. Specific message: Replicate *replicate_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The cdr delete replicate command was run.
Class ID: 69 Event ID: 69001	Class message: A replication state change event has happened. Specific message: Replicate set *replicateset_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The cdr delete replicateset command was run.
Class ID: 70 Event ID: 70001	Class message: A replication state change event has happened. Specific message: Server *server_name* is modified.	3	This event alarm is disabled by default. The cdr modify server command was run.
Class ID: 71 Event ID:	Class message: ER: Network connection disconnected. Specific message:	3	The connection was closed as the result of an Enterprise Replication command, such as cdr stop, cdr disconnect server, or cdr delete server. This event alarm appears on the database server on which the command was run and might or might not appear on the peer server. The peer server might receive

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
71001	Network connection was dropped from the server *server_name* to the server *server_name*. Connection closed due to an Enterprise Replication administrative activity.		event alarm 71 with the specific message that the connection closed for an unknown reason because the administrative control message might not reach the peer server before the connection is closed. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and that an administrative command was run. **ER State:** How replication is affected and how to reestablish the connection depends on which command closed the connection. • If the cdr stop command was run, replicated transactions are no longer being captured from this database server. • If the cdr disconnect server command was run, replicated transactions continue to be captured and queued. • If the cdr delete server command was run, the database server is no longer a participant in the replication domain and no replicated data is captured on or for this database server. **User Action:** Solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.
Class ID: 71 Event ID: 71002	Class message: ER: Network connection disconnected. Specific message: Network connection was dropped from the server *server_name* to the server *server_name*. Connection	3	An idle timeout occurs when there are no replication messages sent between the replication servers for the number of seconds specified as the idle timeout period. The connection is reestablished automatically when replication messages are ready to be sent. This event alarm appears on both database servers affected by the dropped connection.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
	closed due to the idle time-out set for the replication server.		**Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and the reason of idle timeout. **ER State:** Replication continues normally. **User Action:** None. Replication resumes automatically. You can increase or eliminate the idle timeout period by using the cdr modify server command.
Class ID: 71 Event ID: 71003	Class message: ER: Network connection disconnected. Specific message: Network connection was dropped from the server *server_name* to the server *server_name*. Connection unexpectedly closed for an unknown reason.	3	This event alarm occurs when there is a connection problem not related to Enterprise Replication, such as a network outage or one of the database servers shutting down. This event alarm might appear on both database servers affected by the dropped connection. This alarm does not appear on a database server that shut down. This alarm might appear when a peer server closed the connection with an administrative activity, in which case that server receives event alarm 71 with the specific message that an administrative activity closed the connection. **Online Log:** A message appears stating: CDR connection to server lost, with the server ID, server name. **ER State:** Replicated transactions continue to be captured and queued, except on database servers that are shut down. Replicated transactions to and from the affected servers are not transmitted. **User Action:** Examine both servers to determine the cause of the dropped connection. • If there was a network problem, solve it and restart any database servers that might be shut down. The Enterprise Replication connection reconnects automatically.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
			- If there was an administrative action, solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.
Class ID: 73 Event ID: 73001	Class message: Enterprise replication NIF connection terminated. Specific message: Enterprise Replication: Connection to *server_name* closed. Reason: CDR server *server_name* not found.	3	The remote server initiated a connection request that was not completed by the local server. This alarm appears when the remote server has an `sqlhosts` entry for the local server, but the local server does not have a corresponding `sqlhosts` entry for the remote server. This situation can occur when a new server is added to the domain but the local server does not have an entry for that server in its `sqlhosts` file. **Online Log:** The specific message also appears in the online log. **ER State:** The new server cannot participate in replication until the `sqlhosts` entries are correct. Replication between the established replication servers continues normally. **User Action:** To solve this issue, update the `sqlhosts` entry on the local server with the appropriate entry for the remote server. Make sure that all the `sqlhosts` files are consistent on all replication servers in domain.
Class ID: 74 Event ID: 74001	Class message: Enterprise replication recovery failed Specific message: Server name/id mismatch in sqlhosts file while recovery, recovered name =	3	The server information in the `sqlhosts` file was updated after the server was defined for replication. This alarm can appear after the following sequence of events:

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
	server_name, id = *ID*, current name = *server_name*, id = *ID*		1. Replication is stopped on a server because the cdr stop command was run or the server was shut down. 2. The `sqlhosts` file was updated. 3. Replication was attempted to be restarted by running the cdr start command or by starting the server. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Update the `sqlhosts` file to restore the original server information and then restart replication by running the cdr start command or restarting the database server.
Class ID: 75 Event ID: 75001	Class message: ER: the logical log replay position is not valid. Restart ER with the cdr cleanstart command, and then synchronize the data with the cdr check --repair command. Specific message: The replay position (logical log ID *log_number* and log position *log_position*) has been overwritten.	4	This event alarm occurs if a database server was shut down or replication was stopped for long enough to fill a logical log. When the database server was restarted or the cdr start command was run, replication failed. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the cdr cleanstart command and then synchronize the data by running the cdr check replicateset command with the --repair option.
Class ID: 75 Event ID: 75002	Class message: ER: the logical log replay position is not valid. Restart ER with the cdr cleanstart command, and then synchronize the data with the cdr check --repair command.	4	This event alarm can occur after a point in time restore was performed on the database server. The point in time restore applied log records beyond the current replay position. **Online Log:** The specific message also appears in the online log.

Table 21. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
	Specific message: The replay position (logical log ID *log_number* and log position *log_position*) is later than the current position.		**ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the cdr cleanstart command and then synchronize the data by running the cdr check replicateset command with the --repair option.
Class ID: 76 Event ID: 76001	Class message: A replication state change event has happened. Specific message: Server *server_name* is disabled.	3	This event alarm is disabled by default. The cdr disable server command was run.
Class ID: 77 Event ID: 77001	Class message: A replication state change event has happened. Specific message: Server *server_name* is enabled.	3	This event alarm is disabled by default. The cdr enable server command or the cdr check replicateset command with the --enable option was run.

Related reference

cdr delete replicate on page 374

cdr delete replicateset on page 376

cdr delete server on page 377

cdr change replicate on page 299

cdr change replicateset on page 302

cdr connect server on page 337

cdr start on page 443

cdr stop on page 465

cdr suspend server on page 473

cdr resume server on page 442

cdr delete template on page 381

cdr suspend replicate on page 471

cdr suspend replicateset on page 472

cdr resume replicate on page 440

cdr resume replicateset on page 441

cdr start replicate on page 445

cdr start replicateset on page 448

cdr stop replicate on page 468

cdr stop replicateset on page 469

cdr modify replicate on page 415

cdr modify replicateset on page 419

cdr modify server on page 421

Related information

Monitor and troubleshooting Enterprise Replication on page 197

Enabling or Disabling Enterprise Replication Event Alarms

You can control which Enterprise Replication event alarms can be raised.

About this task

By default, Enterprise Replication event alarms are enabled, except most of the state change event alarms that are raised by specific **cdr** commands. You might want to enable state change event alarms to track which **cdr** commands are being run, but if you are setting up your replication system and running many **cdr** commands, the resulting large number of event alarms might affect system performance.

To change which Enterprise Replication event alarms are enabled, reset the values of the CDR_ENV configuration parameter for the CDR_ALARMS environment variable:

1. Add a new line or update the existing line for CDR_ENV CDR_ALARMS in the `onconfig` file. List all the Enterprise Replication event alarms that you want to be enabled.
2. Restart the database server.

Example

Example

The following example shows the CDR_ENV value in the `onconfig` file for the CDR_ALARMS environment variable with event alarm 49 enabled in addition to the default event alarms:

```
CDR_ENV CDR_ALARMS=30-39,47-50,71,73-75
```

Related reference

CDR_ALARMS Environment Variable on page 520

CDR_ENV Configuration Parameter on page 502

Push data feature

Push data feature lets clients register for changes in a dataset using simple SELECT statements and WHERE clauses. Once the server captures data for push data event conditions which evaluates to true for WHERE clause condition, the server pushes committed data to the client, based on registered events. Scaling is achieved by clients not having to poll for data, and not having to parse, prepare, and execute SQL queries. Database servers with parallel architecture -- Enterprise Replication log snooper and grouper -- feed the data to all clients by asynchronously reading logical log file changes. This design lets client applications scale linearly without adding significant overhead to the database server or any OLTP applications making changes to the database. Data that is returned to the client is in a developer-friendly JSON format.

Table 22.

Input attribute name	Description
table	Table name to be registered
owner	Table owner
database	Database name
query	SELECT statement including projection list and WHERE clause to register for changes in a data set.
label	User defined string to be returned along with an event document. This attribute is useful to differentiate between events when more than one push-data event is registered within the same session
timeout	The amount of time a client is blocked in the smartblob read API for an event data. The server returns timeout json document when a timeout condition is triggered. Supported range of values are: • -1 to wait forever • >=0 to wait for a specified amount of time in seconds.
commit_time	Returns event data that is committed after the stated transaction commit time.
txnid	A unique 8 byte ID:

Table 22.

(continued)

Input attribute name	Description
	• Higher order 4 bytes: commit work log ID • Lower order 4 bytes: commit work log position
max_pending_ops	Maximum number of event records to be kept in the pending session .
maxrecs	Maximum number of records to be returned by the smartblob API read call.

Grant replication permission on sysadmin database for the user registering push data events:

```
execute function task('grant admin', 'user1', 'replication');
```

Register client as a push data session by using the sysadmin task command:

```
execute function informix.task('pushdata open')
```

The above command registers the client as a push-data session, and returns a unique session ID. This ID is needed for reading event documents using the smartblob readAPI.

This command also auto-registers enterprise replication, when it has not been defined earlier.

To internally define enterprise replication automatically, the pushdata open command relies on the existence of at least one storagepool entry to create the dbspace and subspace required for defining enterprise replication. You must create a storagepool entry using the task API.

For example:

```
Execute function task( 'storagepool add', '/informix/storage', '0', '0', '20000', '1' );
```

Registering one or more push data event conditions using the sysadmin task command:

```
execute function informix.task('pushdata register',
 {table:"creditcardtxns",owner:"informix",database:"creditdb",query:"select uid, cardid,
 carddata from creditcardtxns where carddata.Amount::int >= 100",label:"card txn alert"})
```

Registering session-specific attributes, like timeout, using the pushdata register task command:

```
execute function informix.task('pushdata register', { timeout:"60",max_pending_ops:"0",maxrecs:"1"})
```

De-registering one or more registered event conditions using the pushdata deregister command:

```
To de-register one or more event conditions for the given table:
execute function informix.task('pushdata deregister', {table:"usertable",owner:"informix",database:"ycsb"})
```

To de-register all event conditions with the same label attribute tag:

```
execute function informix.task('pushdata deregister', { label:"card txns"})
```

Note: To deregister a specific event condition, either use the label attribute, or specify a query attribute, along with the table, owner and database attributes.

API to read event data:

The client must invoke the smartblob read API to read an event data. Input for the smartblob read API must include:

- The session ID returned from running the pushdata open task command.
- The input buffer pointer
- The input buffer size-this should be at least equal to the sum of the before image size, the after image size, and 1024 bytes. If multiple records are expected from one read call, then the input buffer size should be equal to the sum of the before image size, the after image size, and 1024, multiplied by the number of records.
- The error code pointer.

ESQLC READ API Example:

```
/*
 * Read data into the buffer
 */
 bytesread = ifx_lo_read(sessionid, databuf, bytes_per_read,loreaderr);
```

Table 23.

Attribute name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner
database	Database name
label	Optional user-specified data for the event condition
txnid	8 byte unique ID: • higher order 4 bytes: commit work log ID • lower order 4 bytes: commit work log position.
operation_owner_id	User id of the user executing the IUD operation.
operation_session_id	Session id of the session executing the IUD operation

Table 23.

(continued)

Attribute name	Description
commit_time	Transaction commit time for the event data.
op_num	Increasing sequence number for the event document within a given transaction. If the transaction generates 10 events, then each document returned will have an incremental op_num value, starting from 1 to 10.
restart_logid	Restart (replay) position logical log unique id. This position may be used to reset Enterprise replication capture position upon server failure using ‘ "pushdata reset_capture’ ADMIN/TASK API.
restart_logpos	Restart (replay) position logical log position within the given log unique id. This position may be used to reset Enterprise replication capture position upon server failure using ‘ "pushdata reset_capture’ ADMIN/TASK API.
rowdata	Row data in JSON document format. Data is returned using the column name as key and the column data as value.
before_rowdata	Before row data for an Update operation.
ifx_isTimeout	Document with this attribute is returned with its value set to true if no event gets triggered within the timeout interval that is specified by the client.
ifx_warn_total_skipcount	A warning document with this attribute is returned, containing the cumulative number of events that are discarded, from exceeding the max_pending_ops attribute threshold.

Sample output from the smartblob read API for an Insert operation:

```
{“operation?:"insert",?table?:"creditcardtxns",?owner?:"informix",
?database?:"creditdb",?label?:"card txn
 alert",?txnid?:2250573177224,?operation_owner_id?:201,?operation_session_id?:200,
?commit_time?:1488243530,?op_num?:1,?restart_logid?:31,?restart_logpos?:24,?rowdata?:{“uid?:22,
?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams
 Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}
```

Sample output from the smartblob read API for Update operation:

```
{“opertion?:"update",table:"creditcardtxns",?owner?:"informix",?database
?:"creditdb",?label?:"card txn
 alert",?txnid?:2250573308360, ?operation_owner_id?:201,?operation_session_id?:205,
?
commit_time?:1488243832,?op_num?:1,?restart_logid?:31,?restart_logpos?:24,?rowdata?:{uid:21,cardid:"7777-7777-7
777-7777",
?carddata?:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },?before_rowdata?:{“uid?:21,
?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z
  } }}
```

Sample output from the smartblob read API for Delete operation:

```
{“opertion?:"delete",?table?:"creditcardtxns",?owner?:"informix",?database?:"creditdb",?label?:"card txn
 alert",
?txnid?:2250573287760,?operation_owner_id?:201,
?operation_session_id?:209,?commit_time?:1488243797,?op_num?:1,?restart_logid?:31,?restart_logpos?:24,
?rowdata?:{“uid?:22,?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams
 Club","Amount":200,"Date":2017-05-01T13:35:06.000Z } }}
```

Sample output from the smartblob read API for a multi-row buffer, when the maxrecs input attribute is set to greater than 1:

```
{[
{“operation?:"insert",?table?:"creditcardtxns",?owner?:"informix",?database?:"creditdb",?label?:"card txn
 alert",
“txnid?:2250573309999, ,?
operation_owner_id?:201,?operation_session_id?:212,?commit_time?:1487781325,?op_num?:1,
?rowdata?:{uid:"7",?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams
 Club","Amount":200,"Date":2017-05-01T15:10:10.000Z } }},
{“operation?:"insert",table:"creditcardtxns",?owner?:"informix",?database?:"creditdb",?label?:"card txn
 alert",?txnid?:2250573177224,
?
operation_owner_id?:201,?operation_session_id?:215,?commit_time?:1488243530,?op_num?:1,?restart_logid?:31,?rest
art_logpos?:24,
?rowdata?:{“uid?:22,?cardid?:"6666-6666-6666-6666",?carddata?:{"Merchant":"Sams
 Club","Amount":200,"Date":2017-05-01T16:20:10.000Z } }}
]}
```

Using the sample *pushdata* ESQL/C program

You can run the *pushdata* ESQL/C program to safely preview the process of registering event triggers with your Informix server, to retrieve event data in JSON format.

The program file, **pushdata.ec**, can be found in the *$INFORMIXDIR/demo/cdc* folder of your Informix installation folder.

Push data session survival

By default, if client gets disconnected from server, server deletes push-data event conditions for the session, and server no longer capture the event data. The following APIs allow server to capture the event data for the push-data client if client is temporarily disconnected from the server.

Detachable sessions can survive from server failure. Client can re-attach to detachable sessions after bouncing the server or after primary server failover operation in a cluster environment with secondary servers. After bouncing server, it is possible for the pushdata client to get duplicate records. Client need to save last received record txnid and op_num and discard the processed records. *restart_logid* and *restart_logpos* are added to INSER, UPDATE and DELETE records returned to the push data sessions. After server failover or upon restarting Informix server, client can re-attach to the detached session and reset Enterprise replication capture position back to *restart_logid* and *restart_logpos* using ‘pushdata reset_capture’ sysadmin task/admin API.

Sample record for Insert operation:

```
{"operation":"insert","table":"t1","owner":"informix","database":"testdb","txnid":133151498608,
"operation_owner_id":37188,"operation_session_id":67,"commit_time":1568823415,"op_num":4,"restart_logid":31,
"restart_logpos":24,"rowdata":{"c1":4001,"c2":4001 }}
```

- To mark a session as a detachable session:

  ```
  execute function informix.task('pushdata setdetach') into :retvalstr;
  ```

  This API returns unique session id.
- Optional command to recapture data from a previous log position after bouncing server or primary server failover operation:

  ```
  execute function admin("pushdata reset_capture", '{"logid?:?%d?, "logpos?:?%d?}’);
  ```

- This command API stops Enterprise Replication(ER) and restarts ER with the given log position set to replay position. This API impacts all pushdata sessions and existing replicate definitions in Enterprise Replication.

  Note: Care must be taken to set logical log unique id and position to the last known *restart_logid* and *restart_logpos* returned to push data session.

  restart_logid and *restart_logpos* are added to INSERT, UPDATE and DELETE records returned to the push data sessions. After server failover or upon restarting Informix server, user can attach back to the detached session and reset Enterprise replication capture position back to *restart_logid* and *restart_logpos*.
- To re-join a detached session after reconnecting to server:

  ```
  execute function informix.task('pushdata join', "{session_id:"245"}) into :retvalstr;
  ```

  Returns smartblob file descriptor to read event data. Unique session id returned from 'pushdata setdetach' API is used as input to 'pushdata join' API. 'pushdata join' returns smartblob file descriptor -- same as 'pushdata open'.

  Note: 'pushdata open' and 'pushdata join' APIs should not be called in the same session.

  'pushdata open' is used to establish new push-data session, and 'pushdata join' API is used to join existing detached push-data session.
- To delete "detachable" push-data session information from the server:
  - To delete currently attached push-data session:

    ```
    execute function informix.task('pushdata delete') into :retvalstr;
    ```

    This is only required if the push-data session is marked as a detachable session using API 'pushdata setdetach'.
  - To delete a specific unique session id currently not attached to any client application:

    ```
    execute function task('pushdata delete', '{session_id:"12"}') ;
    ```

  - To delete all push-data sessions that are not attached to any client applications:

    ```
    execute function task('pushdata delete', '{delete_all:"1"}');
    ```

Detach trigger

Using the detach trigger methods in IfmxThreadedSmartTrigger, you can declare a Smart Trigger to be 'detachable'. A detachable trigger has an unique identifier which allows you to reconnect to the session on the server.

```
/* Detach a trigger */
IfxSmartTrigger push = new IfxSmartTrigger("jdbc-url-here");
push.detachable(true); //Set the trigger as detachable
push.open();
String session1 = push.getDetachableSessionID(); //Get the session id
//Closes the JDBC connection and returns the session ID
//This is the same session id as you get from the call above
Session1 = push.detach();
```

On detaching from the session, you can create a new Smart Trigger object and pass in the session ID.

```
push = new IfxSmartTrigger(jdbc-url-here);
//Assign the session ID before you start the smart trigger
push.sessionID(sessionID);
TestPushCallback callback1 = new TestPushCallback();
push.registerCallback("test-label-pushtest", callback1);
push.start();
//You pick up where you left off, retrieving any messages you missed from the server
```

Loopback Replication

You can setup replication between tables within the same Informix server with loopback replication. Source and target tables can be in the same database or on two different databases within the same server.

This section discusses the intended audience and the associated software products that you must have to use Enterprise Replication.

Loopback Configuration

To setup loopback replication, define pseudo ER group in SQLHOSTS, and define new service within DBSERVERALIAS other than the one used for primary ER group name. Service name added for loopback pseudo group should appear after service name used for primary Enterprise Replication group.

SQLHOSTS file example:

```
g_er_server group - - i=1
er_server onsoctcp *myhost 17001  g=g_er_server
g_loopback group - - i=2
loopback onsoctcp *myhost 17002  g= g_loopback
```

DBSERVERALIAS example:

```
DBSERVERALIAS er_server,loopback
```

where g_er_server is the primary group name for the local server, g_loopback is the pseudo ER group name for the local server.

Replication definition between primary and pseudo groups

Replicate is defined only using **primary->target configuration**, and replication definition should include only primary group and pseudo group participants.

Note: Only **ignore** and **always apply** conflict resolutions rules are supported for loopback replication.

```
$ cdr define repl --connect=g_er_server test_t1 --conflict=always --scope=row --ats --ris --floatieee
 --master=g_er_server
"P test@g_er_server:informix.t1" "select * from t1" "R test@g_loopback:informix.t2"  "select * from t2"
```

In the above replicate definition is established between table t1 and table2 in test database on the same server

All ER commands including ‘cdr sync’, ‘cdr check’, template commands work with pseudo group. The only restriction is that pseudo ER group cannot be added to grid definition, and grid commands are not supported with pseudo ER group.

Template example

Template can be used to define replication across multiple tables from two different databases.

```
$ cdr define template --connect=g_er_server ifxt_test_test --conflict=always --scope=row --ats --ris
 --floatieee --master=g_er_server --database=test  --all
$ cdr realize template --connect=g_er_server ifxt_test_test "test@g_er_server"
$ cdr realize template --connect=g_loopback ifxt_test_test --target "test2@g_loopback"
```

The above commands define replication between tables in test and test2 databases on the same server

‘cdr list replicate’ command show ‘loopback’ replication attribute

```
REPLICATE:          rep1
STATE:              Active ON:g_informix
CONFLICT:           Always Apply
FREQUENCY:          immediate
QUEUE SIZE:         0
PARTICIPANT:        ycsb:informix.usertable
OPTIONS:            row,ris,ats,fullrow,loopback
REPLID:             65537 / 0x10001
REPLMODE:           PRIMARY  ON:g_informix
APPLY-AS:           INFORMIX ON:g_informix
REPLTYPE:           Master
```

Sample procedure to reorg table online

1. Create RAW table with appropriate fragmentation strategy

   Note: Create standard table if you have secondary servers.

2. Initial data load -- Unload and load data from old table to new table
3. Create indexes
4. Convert table type to “standard"
5. Define replication and repair data using ‘cdr check --repair’ command

   Note: Define replicate with --name=n option, if you plan to rename table while replication is still active.

6. Use rename table DDL or synonym to point applications to new table
7. Drop replicate once queues are empty

Sample procedure for online migration of database codeset

1. Create new database with target codeset and schema
2. Unload data from old database using appropriate CLIENT and DB locales, and load data into new database use CLIENT_LOCALE set to unloaded data codeset format, and DB_LOCALE set to target database codeset
3. Define replication using –utf8 option and execute data re-synchronization task using cdr check –repair or cdr sync replset commands
4. Use rename database DDL to point applications to new database

Appendixes

The cdr utility

You use the cdr utility to configure and control Enterprise Replication from the command line on your UNIX™ or Windows™ operating system.

You must be the Enterprise Replication server administrator to run any of the cdr commands except the cdr list commands, unless otherwise noted.

The cdr utility requires a certain amount of memory resources. If you encounter out-of-memory errors for cdr commands, your operating system limits on memory use might be set too low. For example, you can run the ulimit command in a UNIX™ environment to configure limits on memory resources. Increase the values for memory resources to avoid out-of-memory errors.

You can run cdr commands from within SQL statements by using the SQL administration API. Most cdr commands that perform actions are supported by the SQL administration API; cdr commands that show information are not supported.

Related information

Repairing Failed Transactions with ATS and RIS Files on page 187

cdr argument: Administer Enterprise Replication (SQL administration API) on page

Interpret the cdr utility syntax

The cdr utility uses specific terminology and conventions.

Each cdr command follows the same approximate format, with the following components:

- Command and its variation

  The command specifies the action that is taken.

- Options

  The options modify the action of the command. Each option starts with a minus (-) or a double-minus (--).

- Target

The target specifies the Enterprise Replication object that is acted upon.

- Other objects

  Other objects specify objects that are affected by the change to the target.

If you enter an incorrect cdr command at the command-line prompt, the database server returns a usage message that summarizes the cdr commands. For a more detailed usage message, enter cdr *variation* -h. For example, cdr list server -h.

Related information

Enterprise Replication Server administrator on page 18

Command Abbreviations

For most commands, each of the words that make up a **cdr** command variation can be abbreviated to three or more characters.

For example, the following commands are all equivalent:

```
cdr define replicate
cdr define repl
cdr def rep
```

The exceptions to this rule are the **replicateset** commands, which can be abbreviated to **replset**. For example, the following commands are equivalent:

```
cdr define replicateset
cdr def replset
```

Command abbreviations are not allowed when you run **cdr** commands within SQL statements using the SQL administration API. For more information, see the *HCL® Informix® Administrator's Reference*.

Option Abbreviations

Each option for a **cdr** command has a long form and a short form. You can use either form, and you can mix long and short forms within a single command.

On UNIX™, a long form example might look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=/cdr/ats --initial utah
```

On WINDOWS, the same long form example would look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=D:\cdr\ats --initial utah
```

Using short forms, you can write the previous examples as follows:

UNIX™:

```
cdr def ser -c ohio -i 500 -A /cdr/ats -I utah
```

WINDOWS:

```
cdr def ser -c ohio -i 500 -A D:\cdr\ats -I utah
```

The long form is always preceded by a double minus (--). Most (but not all) long forms require an equal sign (=) between the option and its argument. The short form is preceded by a single minus (-) and is usually the first letter of the long form. The short form never requires an equal sign. However, sometimes the short form is capitalized and sometimes it is not. To find the correct syntax for the short form, check the table that accompanies each command variation.

ℹ **Tip:** Use the long forms of options to increase readability.

With the exception of the keyword **transaction**, all keywords (or letter combinations) that modify the command options must be written as shown in the syntax diagrams. For example, in the Conflict Options on page 346, the option **conflict** can be abbreviated, but the keyword **ignore** cannot be abbreviated. Both of the following forms are correct:

```
--conflict=ignore
-C ignore
```

Option Order

You can specify the options of the **cdr** commands in any order. Some of the syntax diagrams show the options in a specific order because it makes the diagram easier to read.

Do not repeat any options. The following fragment is incorrect because **-c** appears twice. In most cases, if you duplicate an option you will receive an error. However, if no error is given, the database server uses the last instance of the option. In the following example, the database server uses the value `-c utah`:

```
-c ohio -i 500 -c utah
```

ℹ **Tip:** For ease of maintenance, always use the same order for your options.

Long Command-Line Examples

The examples in this guide use the convention of a backslash (\) to indicate that a long command line continues on the next line.

The following two commands are equivalent. The first command is too long to fit on a single line, so it is continued on the next line. The second example, which uses short forms for the options, fits on one line.

On UNIX™, the command line might look like:

```
cdr define server --connect=katmandu --idle=500 \
   --ats=/cdrfiles/ats

cdr def ser -c katmandu -i 500 -A /cdrfiles/ats
```

On Windows™, these command lines might look like:

```
cdr define server --connect=honolulu --idle=500 \
   --ats=D:\cdrfiles\ats

cdr def ser -c honolulu -i 500 -A D:\cdr\ats
```

For information on how to manage long lines at your command prompt, check your operating system documentation.

Long Identifiers

Identifier names used in **cdr** commands follow the guidelines of SQL syntax.

Identifiers are the names of objects, such as database servers, databases, columns, replicates, replicate sets, and so on, that Informix® and Enterprise Replication use.

An identifier is a character string that must start with a letter or an underscore. The remaining characters can be letters, numbers, or underscores. On HCL Informix®, all identifiers, including replicates and replicate sets, can be 128 bytes long. However, if you have any database servers in your replication environment that are an earlier version, you must follow the length restrictions for that version.

For more information about identifiers, see the *HCL® Informix® Guide to SQL: Syntax*.

The length of a path and file name, such as the names of ATS files, can be 256 bytes.

User login IDs can be a maximum of 32 bytes. The owner of a table is derived from a user ID and is thus limited to 32 bytes.

Connect Option

Most **cdr** commands allow a connect option to specify the database server to connect to for performing the command.

The **--connect** option causes the command to use the global catalog that is on the specified server. If you do not specify this option, the connection defaults to the database server specified by the **INFORMIXSERVER** environment variable.

Connect Option

“{ -c *server* | --connect=*server* | -c *server_group* | --connect=*server_group* } ”

Element	Purpose	Restrictions	Syntax
server	Name of the database server to connect to	The name must be the name of a database server or server connection.	Long Identifiers on page 256
server_group	Name of the database server group that includes the database server to connect to	The name must be the name of an existing database server group.	Long Identifiers on page 256

You must use the **--connect** option when you add a database server to your replication environment with the **cdr define server** command.

You might use the **--connect** option if the database server to which you would normally attach is unavailable.

If your replication domain contains database servers that are running different server versions, cdr commands must connect to the server running the latest version of HCL Informix®. If you are connected to a database server running an older version of HCL Informix®, you cannot run a cdr command on a database server running a later version of HCL Informix®.

If the database server uses trusted connections between replication servers by including the s=6 option in the `sqlhosts` entries, you configure a regular connection to an alias of the server for the cdr utility to use. In a trusted connection environment, the cdr utility can only connect to the local replication server.

Related information

Enterprise Replication Terminology on page 3

Creating sqlhost group entries for replication servers on page 58

The onpassword utility on page

Connect to another replication server on page 164

Configuring secure ports for connections between replication servers on page 59

Participant and participant modifier

A participant defines the data (database, table, and columns) to be replicated on a specific database server. You can choose whether to allow the participant to both send and receive replicated data, or to only receive replicated data. You can choose whether to allow the participant to both send and receive replicated data, or to only receive or only send replicated data. You can choose to check for table owner permissions when applying operations. By default, permissions are not checked. The participant modifier is a restricted SELECT statement that specifies the rows and columns that are replicated.

Syntax

Participant

“ " [{ P | R | S }] [{ O | I }] *database* @ *server_group* : *owner* . *table* " ”

Participant Modifier

“ " SELECT { { | *column* } | * } FROM *table* *WHERE_Clause* " ”

Element	Purpose	Restrictions	Syntax
column	Name of a column in the table that is specified by the participant.	The column must exist.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
	The replication key columns must be included.		
database	Name of the database that includes the table to be replicated.	The database server must be registered with Enterprise Replication.	Long Identifiers on page 256
owner	User ID of the owner of the table to be replicated.		Long Identifiers on page 256
server_group	Name of the database server group that includes the server to connect to.	The database server group name must be the name of an existing Enterprise Replication server group in the `sqlhosts` information and must be used only once in the same replicate.	Long Identifiers on page 256
table	Name of the table to be replicated. Must be the same table name in the participant and participant modifier.	The table must be an actual table. It cannot be a synonym or a view.	Long Identifiers on page 256
WHERE_Clause	Clause that specifies a subset of table rows to be replicated.	Can include opaque user-defined types that are always stored in row. Cannot contain a column of a **TimeSeries** data type.	WHERE Clause of SELECT on page

The following table describes the participant options.

Option	Meaning
I	Default. Disables the table-owner option (O).
O	Enables permission checks for table owner that is specified in the participant to be applied to the operation (such as INSERT or UPDATE) that is to be replicated and to all actions fired by any triggers. When the O option is omitted, all operations are run with the privileges of user **informix**. Enables permission checks for table owner that is specified in the participant to be applied to the operation (such as INSERT or UPDATE) that is to be replicated and to all actions fired by any triggers. When the O option is omitted, all operations are run with the privileges of user **informix** or the server owner.

Option	Meaning
	On UNIX™, if a trigger requires any system-level commands (as specified by the system() command in an SPL statement), the system-level commands are run as the table owner, if the participant includes the O option. On Windows™, if a trigger requires any system-level commands, the system-level commands are run as a less privileged user because you cannot impersonate another user without having the password, whether the participant includes the O option.
P	For primary-target replicates, specifies that the participant is a primary participant, which both sends and receives replicated data. Do not use for an update-anywhere replicate. Enterprise Replication defines all the participant as primary in an update-anywhere replication system.
R	For primary-target replicates, specifies that the participant is a receive-only target participant, which only receives data from primary participants.
S	For primary-target replicates, specifies that the participant is a send-only primary participant, which only sends data to target participants. You cannot use this option for replicates that include **TimeSeries** columns.

Usage

Each participant in a replicate must specify a different database server. The participant definition includes the following information:

- Database in which the table is located
- Table name
- Table owner
- Participant type
- Participant modifier with a SELECT statement

You must include the server group, database, table owner, and table name when you define a participant, and enclose the entire participant definition in quotation marks.

If you use a SELECT * FROM *table_name* statement, the tables must be identical on all database servers that are defined for the replicate.

If you use a SELECT * FROM *table_name* statement, the tables must be identical on all database servers that are defined for the replicate, unless you implement a data consolidation system by defining one server to receive data and several other servers that only send data.

Restriction: Do not create more than one replicate definition for each row and column combination to replicate. If the participant overlaps, Enterprise Replication attempts to insert duplicate values during replication.

You can define participants with the following commands:

- cdr define replicate
- cdr modify replicate
- cdr change replicate
- cdr define template

The following restrictions apply to a SELECT statement that is used as a participant modifier:

- The statement cannot include a join or a subquery.
- The statement cannot run operations on the selected columns.
- The statement cannot exceed 15 000 ASCII characters in length.
- For tables that have **TimeSeries** columns, all columns must be included.

Replicate only between like data types. For example, do not replicate between the following combinations of data types:

- CHAR(40) to CHAR(20)
- INT to FLOAT

You can replicate between the following types with caution:

- SERIAL and INT
- BYTE and TEXT
- BLOB and CLOB

Note: The ERKEY shadow columns are not included in the participant definition if you use SELECT * in your participant modifier. To include the ERKEY shadow columns in the participant definition, use the --erkey option with the cdr define replicate, cdr change replicate, or cdr remaster commands.

Example

Example 1: Defining update-anywhere participants

If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere by default. For example:

```
"
db1@g_hawaii:informix.mfct"
 "
select * from mfct"
 \
"
db2@g_maui:informix.mfct"
```

```
 "
select * from mfct"
```

Example

Example 2: Defining a primary server

For example, in the following participant definition, the P indicates that in this replicate, **hawaii** is a primary server:

```
"
P db1@g_hawaii:informix.mfct"
 "
select * from mfct"
```

If any data in the selected columns changes, that changed data is sent to the secondary servers.

Example

Example 3: Defining a server that only receives data

In the following example, the R indicates that in this replicate, **maui** is a server that only receives data:

```
"&"R db2@g_maui:informix.mfct"
 "
select * from mfct"
```

The specified table and columns receive information that is sent from the primary server. Changes to those columns on **maui** are *not* replicated.

Example

Example 4: Defining a data consolidation system with servers that only send data

To implement a data consolidation system, you can define one server to receive and consolidate the data and configure several other servers that only send data. In the following example, the S options indicate that the **rome**, **tokyo**, **perth**, and **ny** servers can only send data:

```
"db0@london:user.world_sales" "select * from world_sales"\
"S db1@rome:user1.sales_rome" "select * from sales_rome"\
"S db2@tokyo:user2.sales_tokyo" "select * from sales_tokyo"\
"S db3@perth:user3.sales_perth" "select * from sales_perth"\
"S db4@ny:user4.sales_ny" "select * from sales_ny"\
```

The central server, **london**, is a standard replication server without restrictions on sending or receiving data.

Related reference

cdr define replicate on page 342

cdr modify replicate on page 415

cdr change replicate on page 299

cdr define template on page 367

cdr swap shadow on page 475

Related information

Primary-Target Replication System on page 35

Participant definitions on page 100

Data consolidation on page 36

Primary-Target Data Dissemination on page 35

Return Codes for the cdr Utility

If a cdr command encounters an error, the database server returns an error message and a return code value.

The message briefly describes the error. For information about interpreting the return code, use the cdr finderr command.

The following table lists the return codes.

Table 24. Return codes for the cdr utility

Return code	Error text	Explanation
0	Command successful.	
1	A connection does not yet exist for the given server.	A replication server involved in the command is not connected to the server that is running the command. This error code can be returned when a cdr sync or cdr check task cannot switch connections between task participants. **User action:** Establish connections between all necessary replication servers and rerun the command.
3	Table column undefined.	A column name listed in the SELECT statement of the replicate participant definition is not found in the table dictionary. This error code can be returned if a shadow column name is included in the SELECT statement of the replicate definition. **User action:** Correct the SELECT statement of the participant definition.
4	Incompatible server version.	A cdr command originating on a database server running an older version of Informix® attempted to run on a database server running a later version of Informix®.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Run the command from the replication server running the most recent version of Informix®.
5	Unable to connect to server specified.	A replication server involved in the command is not available for one of the following reasons: • The server disconnected from the domain. • Replication is no longer active on the server. • The server is offline. • The --connect option was not used and the **INFORMIXDIR** environment variable for the current server is not set. This error code can be returned if one of the cdr sync or cdr check task participants cannot be accessed or if a task participant became inactive or went offline while a sync or check task is in progress. This error code can be returned if the user running the cdr define replicate or cdr change replicate command does not have Connect privilege on the database specified for the replicated table. **User action:** Check the status of all participating servers and rerun the command when all servers are active.
6	Database does not exist.	The database name specified for the replicate in the command does not exist. **User action:** Verify the spelling of the database names and that they exist on each participant and rerun the command. This error code can be returned if the cdr view command is run and the **sysadmin** database does not exist.
7	Database not logged.	The database specified for the replicate in the command is a non-logging database. Replicated databases must be logged. **User action:** Change the database logging mode to buffered logging and rerun the command.
8	Invalid or mismatched frequency attributes.	The value for the --at or --every option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with valid and correctly formatted frequency values.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
9	A connection already exists for the given server.	This error code can be returned if the cdr connect server command is run for a server that already has an active connection.
10	Invalid replicate set state change.	The replicate set specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The cdr stop replicateset command is run but all replicates in the replicate set are not active. • The cdr start replicateset command is run but all replicates in the replicate set are already active. • The cdr suspend replicateset command is run but all replicates in the replicate set are not active. • The cdr resume replicateset command is run but all replicates in the replicate set are already active. **User action:** Run the cdr list replicateset and cdr list replicate commands to see the status of each replicate.
11	Undefined replicate set.	The specified replicate set does not exist or the replicate set is empty. The replicate set name might be incorrectly specified in the command. **User action:** Rerun the command with the correct replicate set name, or add replicates to the replicate set and then rerun the command.
12	Replicate set name already in use.	The replicate set name specified in the command is already being used. Replicate set names must be unique in the domain. **User action:** Run the cdr list replicateset command to view a list of replicate set names and then rerun the original command with a different replicate set name.
13	Invalid idle time specification.	The value for the --idle option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted value.
14	Invalid operator or specifier.	Both the --ignoredel y and the deletewins options were used in the same command. These options cannot be used together. **User action:** Rerun the command with only one of these options.
15	Invalid length.	The ATS or RIS directory path specified in the command exceeds 256 characters. This error can be returned if the server group name exceeds 127 characters.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Rerun the command with a shorter directory path or server group name.
16	Replicate is not a member of the replicate set.	The replicate specified to be deleted from the replicate set is not a member of the replicate set. **User action:** Run the cdr list replicateset command for the replicate set to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
17	Participants required for operation specified.	One or more of the participants necessary for this command were not specified. This error code is returned if the sync source server or the target server is not defined as a participant for the cdr sync or cdr check task. This error code is also returned if the target participant list is empty. **User action:** Rerun the command with the required participants.
18	Table does not contain primary key.	The table specified in the command does not have a replication key. This error is returned if the cdr sync or cdr check task cannot find the replication key for the table being repaired. **User action:** Add a primary key constraint or the ERKEY shadow columns to the table and rerun the command. If you have another unique index on the replicated table, you can specify to use the columns in that index as the replication key when you define the replicate.
19	Table does not exist.	The table owner name specified in the command is not correct. This error is also returned if the table owner name is not specified for a table in an ANSI database. **User action:** Rerun the command with the correct table owner name.
20	Server already participating in replicate.	The participant specified in the command is already a participant in the replicate.
21	Command timed out	The command timed out while waiting for queue monitoring to complete. **User action:** Check the server and connection status using the cdr list server command and, if needed, rerun the command and specify a longer timeout period.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
22	Primary key not contained in select clause.	The replicate participant SELECT statement did not include the replication key columns. **User action:** Rerun the command including the replication key columns in the participant SELECT statement.
25	Replicate already participating in a replicate set.	The replicate specified to be added to the replicate set is already a member of the replicate set. **User action:** Run the cdr list replicateset command to view a list of replicates in the replicate set.
26	Replicate set operation not permitted on replicate.	The replicate specified to be deleted from the replicate set does not have a valid name. **User action:** Run the cdr list replicateset command to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
28	Replicate name already in use.	The replicate name specified in the command is already being used. Replicate names must be unique in the domain. **User action:** Run the cdr list replicate command to view a list of replicate names and then rerun the original command with a different replicate name.
29	Table does not exist .	The table name specified in the command does not exist. **User action:** Rerun the command with an existing table name.
30	Invalid replicate state change.	The replicate specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The cdr stop replicate command is run but the replicate is not active. • The cdr start replicate command is run but the replicate is already active. • The cdr suspend replicate command is run but the replicate is not active. • The cdr resume replicate command is run but the replicate is already active. **User action:** Run the cdr list replicate command to see the status of the replicate.
31	Undefined replicate.	The replicate name cannot be found in Enterprise Replication catalog tables. The name of the replicate might be incorrectly specified in the command.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Rerun the command with the correct replicate name.
32	sbspace specified for the send/receive queue does not exist	The CDR_QDATA_SBSPACE configuration parameter is not set to a valid sbspace name. **User action:** Set the CDR_QDATA_SBSPACE configuration parameter to a valid sbspace name in the `onconfig` file.
33	Server not participant in replicate/replicate set.	The server name specified in the command is not a participant in the replicate or replicate set. This error is returned if a server name is not valid. **User action:** To see a list of all participants for each replicate, query the **syscdrpart** view in the **sysmaster** database.
35	Server not defined in sqlhosts.	The server group name specified in the command is not defined in the `sqlhosts` file specified by the **INFORMIXSQLHOSTS** environment variable. **User action:** Check the `sqlhosts` file for the correct spelling of the server group name, or, if necessary, update the `sqlhosts` file to add the server group, and then rerun the original command.
37	Undefined server.	The target participant cannot be found in the Enterprise Replication catalog tables. The name of the server might be incorrectly specified in the command. **User action:** Rerun the command with the correct server name.
38	SPL routine does not exist.	The SPL routine specified with the --conflict option does not exist on one or more participants. **User action:** Make sure that the SPL routine exists on all participants and rerun the command.
39	Invalid select syntax.	The SELECT statement included in the command is not valid or is missing from the command. **User action:** Rerun the command with the correct SELECT statement.
40	Unsupported SQL syntax (join, etc.).	The SELECT statement contains syntax that is not supported for replicate participants. Syntax such as subqueries in the WHERE clause or selecting from multiple tables with a JOIN clause is not supported.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Rerun the command with the correct SELECT statement.
41	GLS files required for data conversion are not installed.	The GLS files required for data conversion to or from UTF-8 are not installed. Code set conversion files are installed with the Client SDK and are in the **$INFORMIXDIR/gls/cv9** directory.
42	Invalid time range.	The time range does not have valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted time range.
43	Participants required for specified operation.	The command did not include the required participant information. **User action:** Rerun the command with participant information.
44	Invalid name syntax.	The name of a replicate or server in the command is not valid, for example, the name might be too long. **User action:** Rerun the command with a valid name.
45	Invalid participant.	The participant syntax is not valid. **User action:** Rerun the command with a valid participant syntax.
47	Invalid server.	A connection between the current server and the specified server is not allowed. This error code is returned if a server attempts to connect to a leaf server that has a different parent server. This error code is also returned if the server specified in the cdr repair command does not exist in the Enterprise Replication catalog.
48	Out of memory.	Enterprise Replication cannot allocate enough memory for this command.
49	Maximum number of replicates exceeded.	The maximum number of replicates that can be defined from a particular server is exceeded. **User action:** Rerun the command while connected to a peer replication server.
52	Server name already in use.	A replication server with this group ID exists. **User action:** Run the cdr list server command to see a list of all replication server names and group IDs
53	Duplicate server or replicate.	A replication server or replicate name is listed more than once in the command.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		This error code is returned if the sync source server is also specified as a sync target server or if the same server is listed multiple times as a sync target participant. This error code is returned if the same group name is defined more than once in the `sqlhosts` file. **User action:** Rerun the command specifying each server and replicate one time.
54	Invalid conflict rule specified.	The conflict resolution rule is not correctly specified. This error code is returned for the cdr define replicate or cdr modify replicate command under the following circumstances: • A stored procedure is specified as the conflict resolution rule but the table has user-defined data types or collection data types. • A secondary conflict resolution rule is specified that is not a stored procedure conflict resolution rule. • A secondary conflict resolution rule is specified but the primary conflict resolution rule is not time stamp or delete wins. This error code is returned if the --timestamp option is used in a cdr check command and the replicate specified in the command does not use the time stamp or the delete wins conflict resolution rule. This error code is also returned when the cdr check command includes the --deletewins option but the specified replicate does not use the delete wins conflict resolution rule. **User action:** Correct the conflict resolution rule issue and rerun the command.
55	Resolution scope not specified.	The conflict resolution scope (row or transaction) is required for ER to resolve conflicts between replicated transactions. Scope is not required if the conflict resolution rule is ignore, in which case ER does not attempt to resolve conflicts. **User action:** Rerun the command with a conflict resolution scope.
56	Shadow columns do not exist for table.	A conflict resolution rule requires the **cdrtime** and **cdrserver** shadow columns but those columns do not exist in the replicated table. **User action:** Alter the replicated table to add the shadow columns by using the ADD CRCOLS clause and rerun the original command.
57	Error creating delete table.	The delete table corresponding to the replicated table was not created.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Check the server message log file for additional error messages.
58	No conflict resolution rule specified.	A conflict resolution rule was not specified in the command. **User action:** Rerun the command with the --conflict option to specify a conflict resolution rule.
61	User does not have permission to issue command.	The user running this command does not have the DBSA privilege at one of the participants in the command. **User action:** Acquire the DBSA privilege on all participants and rerun the command, or rerun the command as a user that has the DBSA privilege at all participants.
62	Enterprise Replication not active.	The command cannot run because Enterprise Replication is not active on the server. **User action:** Run the cdr list server command to see the status of the server.
63	Enterprise Replication already active.	The command cannot make Enterprise Replication active because ER is already active on the server. **User action:** Run the cdr list server command to see the status of the server.
64	Remote/cyclic synchronization not allowed.	The command to define a replication server was attempted on a remote server. **User action:** Rerun the command on the server that is being defined.
65	Server identifier already in use.	The server group ID is not unique. **User action:** Rerun the command with a unique server group ID.
66	No upper time for prune error.	The ending date value for the error pruning range was not specified. **User action:** Rerun the command with a valid ending date.
67	Error not found for delete or update.	The error sequence number does not exist in the errors table. **User action:** Run the cdr error command to see a list of error sequence numbers and then rerun the command with an existing number.
68	Invalid participant mode.	The participant type value is not valid. **User action:** Rerun the command with a valid participant type.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
69	Conflict mode for replicate not ignore or always apply.	One or more replicate participants specified in the command is defined as receive-only and must use a conflict resolution rule of ignore or always. **User action:** Rerun the command with the --conflict option set to ignore or always.
70	Connect/disconnect to/from same server.	The command attempted to connect the local server to itself. **User action:** Rerun the command with a different server name.
72	Cannot delete server with children.	The command did not delete the hub server because the hub server still has child servers. **User action:** Delete the child servers and then delete the hub server.
75	Request denied on limited server.	The command is not allowed on leaf servers. It is also not allowed on replication servers that are disabled. **User action:** If the server is in disabled mode, wait until the server is active and rerun the command.
77	Could not drop the Enterprise Replication database.	The **syscdr** database was not deleted because a client is accessing it. **User action:** Wait for the client to unlock the **syscdr** database and then rerun the command. If necessary, use the --force option to drop the **syscdr** database if Enterprise Replication was partially deleted.
78	Invalid ATS directory.	The ATS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX™) or NUL (Windows™). **User action:** Rerun the command with a valid ATS directory path.
79	Invalid RIS directory.	The RIS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX™) or NUL (Windows™).

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Rerun the command with a valid RIS directory path.
80	Invalid conflict resolution change.	The conflict resolution rule of a replicate cannot be changed to ignore or from ignore.
84	No sync server.	A synchronization server must be specified if the replication server being defined is a non-root or leaf server. The first server in a replication domain must be a root server. **User action:** Rerun the command with the --sync option.
85	Incorrect participant flags.	The participant type or owner option included in the command was not valid. **User action:** Rerun the command with valid participant options.
86	Conflicting leaf server flags.	The --nonroot and --leaf options cannot be used together. **User action:** Rerun the command with only one of the options.
90	CDR connection to server lost, id *group_id*, name *groupname*. Reason: System clocks off by %d seconds.	The system clock times on the servers differ by more than 900 seconds.
91	Invalid server state change.	The server is already in the state indicated by the command. This error code can be returned if the cdr suspend server command is run on a server that is suspended or if the cdr resume server command is run on a server that is active. **User action:** Run the cdr list server command to see the status of the server.
92	CDR is already defined.	Enterprise Replication is already defined on this server.
93	Enterprise Replication is currently initializing.	Enterprise Replication cannot be stopped on the server because replication is in the process of being initialized. **User action:** Run the cdr list server command to see the status of the server. Rerun the command when replication is active.
94	Enterprise Replication is currently shutting down.	Enterprise Replication cannot be stopped on the server because replication is already in the process of being stopped.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Run the cdr list server command to see the status of the server. If necessary, rerun the command.
99	Invalid options or arguments passed to command.	One or more of the options included with this command are not valid options. **User action:** Rerun the command with valid options.
100	Fatal server error.	The command was not completed because of an unrecoverable error condition.
101	This feature of Enterprise Replication is not yet supported.	One of the participants included with this command is running a version of Informix® that does not support this command. **User action:** Rerun the command with valid options.
102	Root server cannot sync with non root or leaf servers.	The synchronization server must be a root server. The --sync option cannot specify a non-root or leaf server. **User action:** Rerun the command specifying a root server with the --sync option.
103	Invalid server to connect.	A non-root server can connect only to its parent or children servers. **User action:** Rerun the command specifying to connect to the parent or a child server.
105	UDR needed for replication was not found.	A user-defined type listed in the SELECT statement of the participant definition does not have one or more of the streamread(), streamwrite(), or compare() support routines. **User action:** Create the required routines for the user-defined type and rerun the command.
106	Setup necessary for UDR invocation could not be completed.	The set-up process necessary to run the streamread(), streamwrite(), or compare() routine for a user-defined type included in the participant definition failed. **User action:** Check to be sure that the required routines for the user-defined type exist. Create them if necessary and rerun the command.
107	Sbspace specified for the send/receive queue does not exist.	The sbspace specified for the CDR_QDATA_SBSPACE configuration parameter is not a valid name or does not exist. **User action:** Correct the value of the CDR_QDATA_SBSPACE configuration parameter or create the sbspace and rerun the command.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
110	Data types with out of row or multi-representational data are not allowed in a replicate where clause.	A replicate participant WHERE clause cannot include a data type that has out-of-row data, such as, a collection data type, a user-defined type, or a large object type. **User action:** Remove the column with out-of-row data from the participant WHERE clause and rerun the command.
111	Cannot have Full Rows off and use stored procedure conflict resolution.	The stored procedure conflict resolution rule requires full row replication. **User action:** Rerun the command without the --fullrow=n option.
112	The replicate set command could only be partially executed. Please run cdr list replset ReplSetName to check results.	The command was successful on some, but not all, of the replicates in the replicate set. **User action:** Run the cdr list replicate command to see the status of each replicate and run the appropriate command on each of the remaining replicates.
113	Exclusive Replset violation.	The specified replicate is a member of an exclusive replicate set, which requires this operation to be performed for the replicate set instead of for individual replicates. **User action:** Run the equivalent command for the replicate set.
115	The syscdr database is missing.	The **syscdr** database cannot be opened. **User action:** Check server message log file for any additional error messages. If you received this error code after running the cdr delete server --force command, no action is required on the server being deleted. Run the cdr delete server command to delete that server on all peer replication servers in the domain. If you receive this error after running the cdr start command, make sure that Enterprise Replication is defined on the local server, and if necessary, define it by running the cdr define server command.
116	Dbspace indicated by CDR_DBSPACE is invalid.	The dbspace specified as the value of the CDR_DBSPACE configuration parameter does not exist. **User action:** Correct the value of the CDR_DBSPACE configuration parameter or create the dbspace and rerun the command.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
117	Enterprise Replication operation attempted on HDR secondary server.	Enterprise Replication commands are not valid on high-availability secondary servers. **User action:** Rerun the command on a high-availability primary server.
118	SQLHOSTS file has multiple entries either at group ID or group name.	There are multiple group definitions for the same group name in the `sqlhosts` file. **User action:** Update the `sqlhosts` file to make all group entries unique.
119	SQLHOSTS file has a problem with (g=) or (i=) option.	The group name specified in the command is not found in the `sqlhosts` file. **User action:** Rerun the command with a valid group name or update the `sqlhosts` file and then rerun the command.
120	Cannot execute this command while ER is active.	Enterprise Replication cannot be deleted on this server because replication is still active. **User action:** Run the cdr stop command and then rerun the cdr delete server --force command.
121	Master participant not found.	The replication server that is specified as the master server in the command does not exist or is not a participant in the specified replicate. **User action:** Rerun the command with the correct master server name.
122	Attempt to perform invalid operation including shadow replicates.	The replicate specified in the command has shadow replicates, which prevent the command from completing. **User action:** Run the cdr list replicate command to see shadow replicate information. Wait for the shadow replicate to be deleted and then rerun the original command. If you are deleting the replicate, delete the shadow replicate and then rerun the original command.
123	Attempt to include an invalid participant in a shadow replicate.	The command attempted to add a participant to a shadow replicate that does not exist in the primary replicate. **User action:** Rerun the command with a valid participant.
124	Invalid command passed to cdrcmd function.	An argument that is not valid was passed to an internal routine. **User action:** Contact IBM® Software Support.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
125	An error occurred concerning a mastered replicate.	The server specified as the master server in the command is not included as a participant in the replicate. This error code is returned if the mastered dictionary verification fails when adding a participant to a mastered replicate. This error code is returned if Enterprise Replication encounters an internal error during master replicate definition. **User action:** Rerun the command with the master server included in the participant list or check the table dictionary.
126	Invalid template participant.	The same table name was specified more than once in the cdr define template command, or a participant name in the cdr realize template command is not valid. **User action:** Rerun the command with unique table names or with valid participant names.
127	Template name already in use.	A replication template with this name exists. **User action:** Rerun the command with a unique template name.
128	Undefined template.	The template name specified in the command does not exist. **User action:** Rerun the command with an existing template name.
129	Cannot delete specified replset as it is a template.	The replicate set specified in the command is a part of a template and cannot be deleted with this command. **User action:** Run the cdr delete template command to delete a template.
131	Sync server not specified.	The synchronization server specified in the command must be the same server that was specified in the cdr define repair command. **User action:** Rerun the command with the correct synchronization server.
132	Invalid sync server specified. Server is not yet defined in ER topology.	The synchronization server specified in the command is not a replication server. **User action:** Rerun the command with an existing replication server as the synchronization server.
134	Cannot lock the replicated table in exclusive mode.	The command cannot obtain an exclusive lock on the table to set alter mode. **User action:** See the online message log file for other errors.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
	For more information see message log file.	
135	Replicate/table is not in alter mode.	The table specified in the command is not in alter mode and therefore alter mode cannot be turned off.
136	Snoopy sub-component is down.	Alter mode cannot be set because the log capture thread was not active.
137	Mismatch between local table dictionary and master dictionary.	The master dictionary does not match the local participant dictionary. **User action:** Check the replicated table definitions on all participants.
138	Replicates not found for table. For more information see message log file.	Alter mode was not turned off because the replicate definitions for the specified table cannot be found. **User action:** Check the spelling of the table name and rerun the command.
139	Mismatch in replicate names or states. Primary and shadow replicate states must match. See the message log file for more information.	The primary and shadow replicates are not in the same state. **User action:** Run the cdr list replicate command to see the replicate state. When the primary and shadow replicates have the same state, rerun the original command.
140	Primary and shadow replicate participant verification failure.	The primary and shadow replicate information does not match.
141	Table is already in alter mode. For more information see message log file.	Alter mode cannot be turned on because the table is already in alter mode.
142	Classic replicates (no mastered dictionary) defined on the table. See message log file for more information.	One or more non-mastered replicates are defined on the specified table. Alter mode requires mastered replicates.
146	Resynchronize error, job name is already in use.	The job name must be unique. **User action:** Rerun the command with a unique job name.
147	Resynchronize error, specified replicate is a shadow repl.	The replicate specified in the command is a shadow replicate. The operation cannot be performed on a shadow replicate. **User action:** Run the cdr list replicate command to see a list of replicate names and rerun the original command with a primary replicate.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
148	Only either participant list or target server can be given for a define repair command.	Both the target server name and a participant list were included in the command. **User action:** Rerun the command with either a target server name or a participant list.
151	Resynch job can be started or stopped only at the source server.	This command must be run from the server specified as the synchronization data source. **User action:** Rerun the command while connected to the synchronization data source server.
154	The replicate being repaired must be in active state.	The replicate specified in the command cannot be repaired because it is not active. **User action:** Run the cdr list replicate command to see the states of replicates.
156	Cannot perform auto remastering process. Replicate is not defined with column name verification option (-name=y). Perform manual remastering process.	Automatic remastering is not possible for the specified replicate. **User action:** Manually remaster the replicate. For instructions, see Remastering replicates without name verification on page 195.
157	CDR: Cannot verify/block grouper evaluation blocking condition.	The specified table cannot be set in alter mode because the grouper component is not active. **User action:** Run the onstat -g grp and onstat -g ddr commands to check the status of the grouper and log capture.
158	CDR: Cannot unblock grouper evaluation.	Alter mode cannot be turned off for the table because the grouper component is not active. **User action:** Run the onstat -g grp and onstat -g ddr commands to check the status of the grouper and log capture.
159	CDR: Grouper evaluation was already blocked in the same transaction. Commit the previous alter statement then re-execute the current alter statement.	More than one alter statement for replicated tables was included in a single transaction. **User action:** Rerun each alter statement in its own transaction.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
160	The specified table was not found in the database. The table specified is either a view or an internally created cdr system table and replicate cannot be defined on views and internally created cdr system tables.	A table name specified in the command was not found or is not a type of table that can be replicated. **User action:** Rerun the command with valid table names.
161	Specified file to read table participants *filename* could not be opened. Please check. Template could not be defined.	The file name specified in the command does not exist. **User action:** If necessary, create the file. Rerun the command with the correct file path and name for the table list.
162	CDR: Local group name not defined in ATS/RIS file.	The ATS or RIS file content is not in the correct format. The file might be corrupted.
163	Error detected while checking replicate attributes on the given table.	The specified table cannot be set to alter mode because the table does not have any master replicates defined. Alter mode requires master replicates. **User action:** Run the cdr list replicate command to see the replicates that are defined for the table.
164	Cannot repair - ATS/RIS repair failed.	The ATS or RIS file content is not in the correct format. The file might be corrupted.
165	Cannot suspend replicate/replset because of dependent repair jobs.	The replicate or replicate set cannot be suspended until the active repair jobs are complete. **User action:** Wait for the repair jobs to complete. Run the cdr list replicate command to see if the shadow replicates associated with the repair jobs still exist. After the shadow replicates are automatically deleted, rerun the original command.
166	Replicate set does not have any replicates.	The replicate set specified in the commands does not contain replicates. This error code is also returned when no replicates are found for a cdr check repair task when the --allrepl option is used. **User action:** Run the cdr list replicateset command for the replicate set to see its replicates.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
167	Enterprise Replication is not supported in Express Edition server.	Enterprise Replication commands cannot be run on servers running Express Edition.
168	The specified table is actually a view, replicate definition on view is not supported.	Replicates cannot be defined on views. **User action:** Rerun the command specifying standard table names.
169	Cannot realize an empty template/	The template cannot be instantiated because it does not contain any replicates. **User action:** Run the cdr list template command to see if the template contains replicates.
170	Template is not yet defined that does not have any replicates.	The template cannot be instantiated because it is not defined. **User action:** Run the cdr list template command to see the names of defined templates.
171	Classic replicates do not support --verify (-v) and/or --autocreate (-u) options.	The --verify and --autocreate options are valid only for master replicates. **User action:** Verify the replicate definition by running the cdr list replicate command.
172	Checksum libraries not installed.	Enterprise Replication cannot find the checksum function for the cdr check replicate or cdr check replicateset command. This error can occur if a replication server is running a version of Informix® that does not support the cdr check replicate or cdr check replicateset command. This error can also occur if the custom checksum function that is specified by the --checksum option is not installed and registered on all replication servers. **User action:** If the replication server is running Informix® version 10.00, make sure that the checksum routines are registered. On Informix® version 10.00, checksum routines must be registered manually. If you specified a custom checksum function, make sure that it is installed and registered on all replication servers.
173	External Sync shutdown requested.	The synchronization task is not active. This error code is returned when Enterprise Replication is being shut down on a replication server participating in a synchronization task started by the cdr sync or cdr checkcommand.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Run the cdr list server or cdr view servers command to see the status of the participating server and when all servers are active, rerun the original command.
174	External Sync abort required.	The synchronization or repair task did not complete in the timeout period. This error can occur if the replicate being synchronized or the shadow replicate that was created to resynchronize the data is not active at all the participants specified in the command. This error code is also returned when the cdr check replicate or cdr check replicateset command is run with the --enable option and the target server cannot be enabled and repaired in the timeout period. The timeout period is 128 seconds or the value you set with the --timeout option. **User action:** Run the cdr list replicate command to check the replicate status. If all participants are active, try running the command again. If the server was being enabled, run the cdr list server command to check the server status. If all participants are active, try running the command again with an increased timeout value.
175	Sync has received a request to stop.	The synchronization or check command was stopped.
176	Sync attempted on replicate which is not active.	The synchronization or check command was stopped because one of the replicates specified is not active.
178	WARNING: Replicate is not in sync.	The replicate is not in sync. This error can be returned after running cdr check replicate or cdr check replicateset. This error can be returned after running cdr check replicate or cdr check replicateset with the --repair option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers on page 177. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the cdr repair

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		command. If there are no ATS or RIS files at the target participants, rerun the original command with the --inprogress option to control how long check task rechecks inconsistent rows that might be in process of being applied at target servers.
181	Value specified cannot be set in-memory, for more information see message log file.	The specified configuration parameter was not modified for the current session. **User action:** For more information, see the server online message log file.
182	Warning: Value specified was adjusted before setting it in-memory, for more information see message log file.	The value of the configuration parameter specified in the command was adjusted and then the configuration parameter was reset for the current session. **User action:** For more information, see the server online message log file.
183	Operation not supported for the specified onconfig variable.	The specified configuration parameter cannot be dynamically updated while the server is running. **User action:** Edit the `onconfig` file and then shut down and restart the server.
184	onconfig text is specified in wrong format.	The value specified for the configuration parameter is not valid. **User action:** For more information, see the server online message log file.
185	Specified variable is an unsupported or unknown ER onconfig or CDR_ENV variable.	The specified configuration parameter or environment variable is not valid in this command. **User action:** Check the spelling of the configuration parameter or environment variable.
186	Value of onconfig variable cannot be changed when ER is defined.	The specified configuration parameter cannot be changed after Enterprise Replication is initialized. **User action:** For more information, see the server online message log file.
187	Value of onconfig variable cannot be changed when HDR is defined.	The specified configuration parameter cannot be changed while the server is participating in a high-availability cluster.
188	WARNING: The onconfig variable is not modified as the	The value specified for the configuration parameter is the same as its current value for the session.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
	specified value is same as stored in the memory.	
189	Replicate cannot be defined or modified since the participant table is protected using Label Based Access Control.	The table specified in the command is using label-based access control (LBAC), which is not supported with Enterprise Replication. **User action:** Rerun the command with a different table name, or remove LBAC from the table and then rerun the command.
190	Code sets specified by CLIENT_LOCALE and DB_LOCALE must be identical.	The ATS or RIS file repair operation requires that the **CLIENT_LOCALE** and **DB_LOCALE** environment variables be set to the same value. **User action:** Reset the value of one of the environment variables to that it matches the other and then rerun the original command.
191	Cannot determine connection server ID for server.	The command cannot obtain the group ID for the server being connected to.
192	Unable to find or connect to a **syscdr** database at a non-leaf server.	The repair command canot find a root server from which to obtain the Enterprise Replication catalog information.
193	SQL failure due to server resource limitations.	An SQL statement failed with memory or lock resource-related error codes.
194	SQL failure due to loss of network connection to server.	An SQL query failed with a network error.
195	SQL failure.	This error code is returned when a command fails due to an SQL error code other than SQL resource limitations-related error codes.
196	Encountered an SQL error.	The command was stopped because an SQL statement failed.
200	Unexpected Internal Error with cdr check or cdr sync.	An internal UDR routine execution might have returned an unexpected error. **User action:** Look at the additional error messages printed on the screen to get more details about this error.
201	Sync/Check encountered an unexpected column type.	The data type of one of the columns being synchronized or checked cannot be resolved for data comparison.
202	Source and Target do not have the same data type.	Corresponding columns on the source server and the target server have different data types.
203	Data for row or column not found.	Enterprise Replication cannot display the column value of a mismatched column on the screen.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
204	Table could not be found.	The table that is being synchronized or repaired is not found on one of the participants. This error code is also returned if the participant being deleted cannot be found in the Enterprise Replication catalog tables.
205	Undefined server group.	The server group specified in the command was not found in the Enterprise Replication catalog tables. **User action:** Rerun the command with an existing server group name.
206	Template not realized at sync data source.	The template cannot be realized on the specified servers because it is not yet realized on the synchronization server. **User action:** Rerun the cdr realize template command specifying the synchronization source server as a participant.
207	Template already realized at one or more of requested servers.	One or more of the participants specified in the command already has the template instantiated on it. **User action:** Run the cdr list replicate command to check the status of the participants and then rerun the original command with the correct list of participants.
208	Server unknown at remote server.	Information about the local server is not available at the remote server.
209	A byte sequence that is not a valid character in the specified locale was encountered	One or more characters in a name specified in the command is not valid.
210	Parameter passed to command (or internally, routine) is invalid.	An argument specified in the command does not have a valid value.
211	Command is too large to be executed as a background task.	The command specified as a background task exceeded 2048 bytes. **User action:** Rerun the command without the --background option.
212	Sync/Check subtask aborted.	One of the tasks that was being performed in parallel was stopped. **User action:** Check the command output to determine which task was stopped.
213	WARNING: set is not in sync.	At least one of the replicates in the specified replicate set is not in sync. This error can be returned after running cdr check replicateset.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		This error can be returned after running cdr check replicateset with the --repair option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers on page 177. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the cdr repair command. If there are no ATS or RIS files at the target participants, rerun the original command with the --inprogress option to control how long the check task rechecks inconsistent rows that might be in process of being applied at target servers.
214	ER: The logical log replay position is not valid. Restart ER with the cdr cleanstart command, and then synchronize the data with the cdr check --repair command.	Enterprise Replication cannot start because the logical log replay position is not valid. **User action:** Run the cdr cleanstart command and then the cdr check replicateset command with the --repair option.
215	Command failed -- The specified table is an external table. You cannot include an external table in a replicate.	Tables created with the CREATE EXTERNAL TABLE statement cannot be included in a replicate.
217	Error with Quality of Data command.	This error can be returned after running the cdr define qod command if the quality of data master server is already defined. This error can be returned after running the cdr start qod command or the cdr stop qod command if the quality of data master server is not defined or the command was run from a different server. **User action:** If the quality of data master does not exist, run the cdr define qod command and then rerun the original command. If the command was run on a different server, rerun the command from the quality of data master server, as indicated in the error message.
220	A node included in the list is not valid.	The server group specified in the grid command was not found in the Enterprise Replication catalog tables.

Table 24. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
		User action: Rerun the command with an existing server group name.
221	The grid name is not unique.	Grid names must be unique among grids and among replicate sets. **User action:** Run the cdr list grid command to see existing grid names and run the cdr list replicateset command to see existing replicate set names. Rerun the original command with a unique grid name.
222	The grid does not exist.	The grid name specified in the command is not the name of an existing grid. **User action:** Run the cdr list grid command to see existing grid names. Rerun the original command with an existing grid name.
223	grid enable user failed	The user name specified in the command does not exist. **User action:** Rerun the command with an existing user name.
224	grid enable node failed	The server name specified in the command is not the name of an existing replication server. **User action:** Rerun the command with an existing replication server name.
225	sec2er failure	The cdr start sec2er command failed. **User action:** Following the instructions in the command output to perform all necessary prerequisites.
227	Region Command Failed	The cdr define region or cdr delete region command failed. **User action:** Review the specific error message and make the appropriate corrections to the command.
228	Grid Table Command Failed.	The cdr change gridtable command failed. **User action:** Review the specific error message and make the appropriate corrections to the command.

Related reference

cdr finderr on page 391

cdr check queue on page 310

Frequency Options

You can specify the interval between replications or the time of day when replication occurs for a replicate.

Frequency Options

“[{ `--immed` | `--every=`*interval* | `--at=`*time* }]”

Element	Purpose	Restrictions
interval	Time interval for replication	The smallest interval in minutes, in one of the following formats: • The number of minutes, as an integer value 1 - 1966020, inclusive. • The number of hours and minutes separated by a colon. The minimum value is 0:01. The maximum value is 32767:59
time	Specific time for replication	Time is given as a 24-hour clock.

The following table describes the frequency options.

Long Form	Short Form	Meaning
--immed	-i	Default. Replication occurs immediately.
--every=	-e	Replication occurs immediately and repeats at the frequency that is specified by interval.
--at=	-a	Replication occurs at the specified day and time.

Usage

The frequency of replication is a property of a replicate. You can set the frequency of replication for a replicate when you define it or modify it. You can reset the frequency of all replicates in a replicate set when you define or modify a replicate set or define a template. For non-exclusive replicate sets, you can update the frequency of individual replicates separately.

If you do not specify a time, replication occurs immediately.

Important: When you use time-based replication by including the --every or the --at option, replicated transactions are split into multiple transactions and referential integrity is not supported. If you want to replicate data intermittently, you can specify the --immed and then disconnect the servers until you want to replicate the data.

Intervals

The --every=*interval* option specifies the interval between actions. The interval of time between replications can be either of the following formats:

- The number of minutes

  To specify the number of minutes, specify an integer value greater than 0. For example, `-e 60` indicates 60 minutes between replications.

  If you specify the interval of time between replications in minutes, the longest interval is 1966020.
- The number of hours and minutes

  To specify hours and minutes, give the number of hours, followed by a colon, and then the number of minutes. For example, `-e 5:45` indicates 5 hours and 45 minutes between replications.

If you specify the length of time in hours and minutes, the longest interval is 32767:59.

Time of Day

Enterprise Replication always gives the time of day in 24-hour time. For example, 19:30 is 7:30 P.M. Enterprise Replication always gives time as the local time, unless the TZ environment variable is set. However, Enterprise Replication stores times in the global catalog in Greenwich Mean Time (GMT).

The --at=*time* option specifies the day on which replication occurs. The string *time* can have the following formats:

- Day of week

  To specify a specific day of the week, give either the long or short form of the day, followed by a period and then the time. For example, `--at=Sunday.18:40` or `-a Sun.18:40` specifies that replication occurs every Sunday at 6:40 P.M.

  The day of the week is given in the locale of the client. For example, with a French locale, you might have `--at=Lundi.3:30` or `-a Lun.3:30`. The time and day are in the time zone of the server.
- Day of month

  To specify a specific day in the month, give the date, followed by a period, and then the time. For example, `1.3:00` specifies that replication occurs at 3:00 A.M. on the first day of every month.

  The special character `L` represents the last day of the month. For example, `L.17:00` is 5:00 P.M. on the last day of the month.

- Daily

  To specify that replication occurs each day, give only the time. For example, 4:40 specifies that replication occurs every day at 4:40 A.M.

Related reference

cdr change replicateset on page 302

cdr define replicate on page 342

cdr define replicateset on page 355

cdr define template on page 367

cdr modify replicate on page 415

cdr modify replicateset on page 419

cdr add onconfig

The **cdr add onconfig** command adds one or more values to a configuration parameter in the ONCONFIG file.

Syntax

```
cdr add onconfig [ <Connect Option > (explicit id ) ] " parameter namevalue "
```

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to set.	You can add values to the following Enterprise Replication configuration parameters: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPPRESS_ATSRISWARN • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: ◦ CDRSITES_731 ◦ CDRSITES_92X ◦ CDRSITES_10X	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr add onconfig** command to add one or more values to an Enterprise Replication configuration parameter while replication is active. The ONCONFIG file is updated. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example adds an sbspace to the existing list of sbspaces for holding spooled transaction row data:

```
cdr add onconfig "CDR_QDATA_SBSPACE sbspace_11"
```

The following example adds the cdrIDs for two version 7.x servers to the existing list of servers:

```
cdr add onconfig "CDR_ENV CDRSITES_731=1,3"
```

Related reference

cdr change onconfig on page 298

cdr remove onconfig on page 434

Related information

Dynamically Modifying Configuration Parameters for a Replication Server on page 162

Enterprise Replication Server administrator on page 18

cdr alter

The **cdr alter** command puts the specified tables in alter mode.

Syntax

```
cdr alter [ <Connect Option > (explicit id ) ] { --on | --off } database:owner.table
```

Element	**Purpose**	**Restrictions**	**Syntax**
database	The name of the database that contains the table	The database server must be registered with Enterprise Replications.	Long Identifiers on page 256
owner	User ID of the owner of the table		Long Identifiers on page 256
table	Specifies the name of the table to put in alter mode	The table must be an actual table. It cannot be a synonym or a view.	Long Identifiers on page 256

The following table describes the options to **cdr alter**.

Long Form	Short Form	Meaning
--on	**-o**	Sets alter mode on.
--off	**-f**	Unsets alter mode.

Usage

Use this command to place a table in or out of alter mode. Use alter mode when you need to alter an attached fragment to the table or you want to perform other alter operations manually.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example puts **table1** and **table2** in alter mode:

```
cdr alter --on db1:owner1.table1 db2:owner2.table2
```

Related reference

cdr swap shadow on page 475

cdr remaster on page 428

Related information

Alter, rename, or truncate operations during replication on page 188

SQL statements and replication on page 21

Enterprise Replication Server administrator on page 18

cdr autoconfig serv

The cdr autoconfig serv command can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can automatically configure replication.

Syntax

```
cdr autoconfig serv [ <Connect Option > (explicit id ) ] [ { <Source options> <Target options> | <Source options> <Target options> } ]
```

Source options

```
" --sourcehost host --sourceport port "
```

Target options

```
" --targethost host --targetport port "
```

Element	Purpose	Restrictions
host	The name of a database server host.	
port	The port number that is used for communication	

The following table describes the options to cdr autoconfig serv.

Long Form	Short Form	Meaning
--sourcehost	-H	The host of the database server that is sending autoconfiguration information. If --sourcehost and --sourceport are not specified, the database server where the command is run is considered the source database server.
--sourceport	-P	The port that is used by the database server that is sending autoconfiguration information.
--targethost	-h	The host of the database server that is receiving autoconfiguration information.
--targetport	-p	The port that is used by the database server that is receiving autoconfiguration information.

Usage

Run the cdr autoconfig serv command to autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and to autoconfigure replication if you are adding database servers to an Enterprise Replication domain. The CDR_AUTO_DISCOVER configuration parameter must be set to `1` on all database servers that are participating in an Enterprise Replication domain or high-availability cluster, before you can run the cdr autoconfig serv command. A newly installed database severs that is added to an Enterprise Replication domain through the cdr autoconfig serv command must have a configured storage pool.

If the source server is already configured for Enterprise Replication, the command performs the following actions:

1. The source server propagates its trusted-host file to target server.
2. The target server adds entries for itself and all other replication servers to its `sqlhosts` file.
3. The source server updates its `sqlhost` file with entries for the target server.
4. Each replication server updates its `sqlhost` file and trusted-host file with entries for the target server.
5. The target server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
6. The target server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
7. The aborted transactions spooling (ATS) file directory `$INFORMIXDIR/tmp/ats_`*`dbservername`* is created on the target server.
8. The row information spooling (RIS) file directory `$INFORMIXDIR/tmp/ris_`*`dbservername`* is created on the target server.
9. Replication of the domain information in the **syscdr** catalog to the target server starts.

If the source server is not configured for Enterprise Replication, the command performs the additional actions:

1. The source server adds entries for itself to its `sqlhosts` file.
2. The source server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
3. The source server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
4. The aborted transactions spooling (ATS) file directory `$INFORMIXDIR/tmp/ats_`*`dbservername`* is created on the source server.
5. The row information spooling (RIS) file directory `$INFORMIXDIR/tmp/ris_`*`dbservername`* is created on the source server.

The following restrictions apply to the cdr autoconfig serv command:

- All replication servers must be active, or the cdr autoconfig serv command fails.
- Do not run the cdr autoconfig serv command if you have configured trusted-host information, manually, rather than through running the admin() or task() function with the cdr add trustedhost argument.
- Do not run the cdr autoconfig serv command if your replication servers have secure ports configured.
- The cdr autoconfig serv command does not copy `hosts.equiv` information to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter. Run the admin() or task() function with the cdr add trustedhost argument if you must add information from the `hosts.equiv` file to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter.

Database servers are configured serially. Parallel configuration is not supported.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Example 1: Define Enterprise Replication on a database server

For this example, you have one database server that is not configured for Enterprise Replication:

- **server_1** on **host_1**

The following command is run on **server_1**.

```
cdr autoconfig serv
```

The command defines Enterprise Replication on **server_1**.

Example

Example 2: Configure connectivity and ER between two stand-alone servers by using source syntax

For this example, you have two stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**

The following command is run on **server_1**:

```
cdr autoconfig serv -c server_2 --sourcehost host_1 --sourceport 9000
```

The command performs the following actions:

1. The command connects to **server_2**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates the domain data to **server_2**.

Example

Example 3: Configure connectivity and ER between two stand-alone servers using target syntax

For this example, you have two stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**

The following command is run on **server_2**:

```
cdr autoconfig serv -c server_1 --targethost host_2 -targetport 9002
```

The command performs the following actions:

1. The command connects to **server_1**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates the domain data to **server_2**.

Example

Example 4: Configure connectivity and ER between two stand-alone servers

For this example, you have three stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**
- **server_3** on **host_3**

The following commands are run on **server_1**:

```
cdr autoconfig serv --targethost hos_t2 -targetport 9002
cdr autoconfig serv --targethost host_3 -targetport 9003
```

The commands perform the following actions:

1. The first command connects to **server_1**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates its data to **server_2**.
5. The second command connects to **server_1**.
6. Enterprise Replication is defined on **server_3**.
7. **server_1** replicates the domain data to **server_3**.

Related reference

cdr autoconfig serv on page 291

CDR_AUTO_DISCOVER configuration parameter on page 498

Related information

cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API) on page

cdr add trustedhost argument: Add trusted hosts (SQL administration API) on page

Preparing the Network Environment on page 55

Trusted-host information on page

Client/server communication on page

Creating sqlhost group entries for replication servers on page 58

cdr change grid

The cdr change grid command adds or deletes replication servers to or from a grid.

Syntax

cdr change grid [<Connect Option > (explicit id)] *grid_name* { **--add** | **--delete** } *server_group*

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256
server_group	Name of a database server group to add to, or remove from, the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

The following table describes the cdr change grid options.

Long Form	Short Form	Meaning
--add	-a	Add the specified replication servers to the grid.
--delete	-d	Delete the specified replication servers from the grid.

Usage

Use the cdr change grid command to add a new replication server to an existing grid or to remove a replication server from an existing grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example adds two servers to a grid named **grid_1**:

```
cdr change grid grid_1 --add gserv3 gserv4
```

The following example removes a server from a grid named **grid_1**:

```
cdr change grid grid_1 --delete gserv1
```

Related reference

cdr define grid on page 338

cdr list grid on page 391

Related information

Grid maintenance on page 130

Adding a replication server to a grid by cloning on page 132

Enterprise Replication Server administrator on page 18

Adding a replication server to a grid by running cdr change grid on page 132

cdr change gridtable

The cdr change gridtable command has multiple uses. It can verify that specific tables can be used in gird queries and then add the tables to a list of verified tables, and it can delete tables from the list of verified tables.

Syntax

```
cdr change gridtable [ <Connect Option > (explicit id ) ] --grid=grid_name --database=database { --add | --delete }{ --all | table }
```

Element	Purpose	Restrictions	Syntax
database	The name of the database.	Must be in a server that is in the specified grid.	Long Identifiers on page 256
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256
table	The name of the table.	The table cannot be a synonym or a view.	Long Identifiers on page 256

The following table describes the cdr change gridtable options.

Long Form	Short Form	Meaning
--add	-a	Add the specified tables to the grid.
--all	-A	Specifies to add or delete all the tables in the database.
--database=	-D	Specifies the database in which the tables are located.
--delete	-d	Delete the specified tables from the grid.
--grid=	-g	Specifies the grid to which to add or delete tables.

Usage

Use the cdr change gridtable command with the --add option to add one or more tables to an existing grid. You add a table to a grid when you want to include the table in grid queries. System tables are automatically included in the grid. When you add a table to a grid, the cdr change gridtable command ensures that every table with that name has the same schema on every grid server. Every table must have the same columns, column names, and data types. The specified database must use the same locale on every grid server.

Use the cdr change gridtable command with the --delete option to remove one or more tables from an existing grid. The specified tables cannot be included in grid queries.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 228.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example adds all tables in the **stores** database to the grid named **grid1**:

```
cdr change gridtable --grid=grid1 --database=stores --add --all
```

The following example removes the table named **customer** in the **stores** database from the grid named **grid1**:

```
cdr change gridtable --grid=grid1 --database=stores --delete=customer
```

Related reference

cdr remaster gridtable on page 431

Related information

Defining tables for grid queries on page 147

Grid queries on page 145

GRID clause on page

cdr change onconfig

The **cdr change onconfig** command replaces the existing value of a configuration parameter with a new value in the ONCONFIG file.

Syntax

cdr change onconfig [<Connect Option > (explicit id)] " *parameter namevalue* "

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to change.	None. All Enterprise Replication configuration parameters and environment variables can be changed with this command.	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows syntax rules for the specific configuration parameter.

Usage

Use the **cdr change onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set Enterprise Replication environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

Suppose the CDR_SUPPRESS_ATSRISWARN configuration parameter is set to suppress the generation of error and warning messages 1, 2, and 10, so that it appears in the ONCONFIG file as: CDR_SUPPRESS_ATSRISWARN 1,2,10. The following command changes the suppressed error and warning messages to 2, 3, 4, 5, and 7:

```
cdr change onconfig "CDR_SUPPRESS_ATSRISWARN 2-5,7"
```

Suppose the CDR_RMSCALEFACT environment variable is set to the value of 4. The following example sets the number of data sync threads started for each CPU VP to 3:

```
cdr change onconfig "CDR_ENV CDR_RMSCALEFACT=3"
```

Related reference

cdr add onconfig on page 289

cdr remove onconfig on page 434

Related information

Dynamically Modifying Configuration Parameters for a Replication Server on page 162

Enterprise Replication Server administrator on page 18

cdr change replicate

The cdr change replicate command modifies an existing replicate by adding or deleting one or more participants.

Syntax

cdr change replicate [<Connect Option > (explicit id)] { **--add***replicate participantmodifier* | **--delete***replicate participant* } [{ **--verify** | [**--autocreate**] }] [**--erkey**]

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier on page 257
participant	Specifies the database server and table for replication.	The participant must exist.	Participant and participant modifier on page 257
replicate	Name of the replicate to change.	The replicate must exist.	Long Identifiers on page 256

The following table describes the options to cdr change replicate.

Long Form	Short Form	Meaning
--add	-a	Adds participants to a replicate.
--autocreate	-u	For use with master replicates only. Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, then they are created automatically. However, the table cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. **Note:** Tables that are created with the --autocreate option do not automatically include non-replication key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the --autocreate option require the use of these objects you must explicitly create the objects by hand.
--delete	-d	Removes participants from a replicate.
--erkey	-K	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the replicate definition, if the table being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used as the replication key.
--verify	-v	For use with master replicates only. Specifies that the cdr change template command verifies the database, tables, and column data types against the master replicate definition on all listed servers

Usage

Use this command to add or delete a participant from a replicate. You can define a replicate that has zero or one participants, but to be useful, a replicate must have at least two participants. You cannot start and stop replicates that have no participants. All participants for the replicate must be online and the cdr utility must be able to connect to each participant.

Important: Enterprise Replication adds the participant to the replicate in an inactive state, regardless of the state of the replicate. To activate the new participant, run cdr start replicate with the name of the server group. See cdr start replicate on page 445.

When you run the cdr change replicate command, an event alarm with a class ID of 65 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Example 1: Add two participants

The following example adds two participants to the replicate named **repl_1**: **db1@server1:antonio.table** with the modifier `select * from table1`, and **db2@server2:carlo.table2** with the modifier `select * from table2`:

```
cdr change repl -a repl_1 \
   "db1@server1:antonio.table1" "select * from table1" \
   "db2@server2:carlo.table2" "select * from table2"
```

Example

Example 2: Remove two participants

The following example removes the same two participants from replicate **repl_1**:

```
cdr change repl -d repl_1 \
   "db1@server1:antonio.table1" \
   "db2@server2:carlo.table2"
```

Example

Example 3: Add a participant that includes ERKEY shadow columns

The following example adds a participant and includes the ERKEY shadow columns from the table **table1**:

```
cdr change repl -a repl_1 --erkey\
   "db1@server1:antonio.table1" "select * from table1"
```

Related reference

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

Enterprise Replication Event Alarms on page 221

Participant and participant modifier on page 257

Related information

Preparing tables without primary keys on page 79

Enterprise Replication Server administrator on page 18

cdr change replicateset

The cdr change replicateset command changes a replicate set by adding or deleting replicates.

Syntax

```
cdr change replicateset [ <Connect Option > (explicit id ) ] { --add | --delete } repl_set replicate
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set to change.	The replicate set must exist.	Long Identifiers on page 256
replicate	Name of the replicates to add to or delete from the set.	The replicates must exist.	Long Identifiers on page 256

The following table describes the options to cdr change replicateset

Long Form	Short Form	Meaning
--add	-a	Add replicates to a replicate set.
--delete	-d	Remove replicates from a replicate set.

Usage

Use this command to add replicates to a replicate set or to remove replicates from an exclusive or non-exclusive replicate set:

- If you add a replicate to an exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.

  If you remove a replicate from an exclusive replicate set, the replicate retains the properties of the replicate set at the time of removal (not the state the replicate was in when it joined the exclusive replicate set).

  When you add or remove a replicate from an exclusive replicate set that is suspended or that is defined with a frequency interval, Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate.
- If you add or remove a replicate to a non-exclusive replicate set, the replicate retains its individual state and replication frequency settings.

Use this command to add or remove replicates from a grid replicate set. You can only add replicates that were created outside of a grid environment to a grid replicate set if the following conditions are met:

- The participant servers must be the same as the servers in the grid.
- The replicated table schema must be the same among all participants.
- The entire replicated table is replicated. Using a SELECT statement in the participant definition that does not include all the columns in the table or includes a WHERE clause is not allowed.
- The replicate must not belong to an exclusive replicate set.
- The replicate must not include **TimeSeries** columns.

When you run the cdr change replicateset command, an event alarm with a class ID of 66 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example adds the replicates **house** and **barn** to replicate set **building_set**:

```
cdr change replicateset --add building_set house barn
```

The following example removes the replicates **teepee** and **wigwam** from replicate set **favorite_set**:

```
cdr change replset --delete favorite_set teepee wigwam
```

Related reference

cdr define replicate on page 342

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

cdr define replicateset on page 355

Enterprise Replication Event Alarms on page 221

Related information

Suspending a Replicate Set on page 174

Frequency Options on page 287

Adding an existing replicate to a grid replicate set by using cdr change replicateset on page 134

Enterprise Replication Server administrator on page 18

cdr change shardCollection

The cdr change shardCollection command changes the sharding definition that determines which database servers are part of the shard cluster.

Syntax

cdr change shardCollection*definition_name* [<Connect Option> (explicit id)] { **--add** { *ER_group* | *ER_group*"*expression*" } | **--drop** *ER_group* | **[--replace]** { *ER_group* | *ER_group*"*expression*" *ER_group***REMAINDER** } | **--partitions** = *partitions* }

Element	**Description**	**Restrictions**
ER_group	The ER-group name of a database server that receives sharded data.	Must be the ER-group name of an existing database server.
definition_name	The name of the sharding definition that is modified.	Must be the name of an existing definition.
expression	The WHERE-clause expression that is used to select rows or documents by shard key or shard column value.	
partitions	The number of hashing partitions on each shard server.	Must be a positive integer.
REMAINDER	Specifies the database server that receives rows or documents with shard key values that are not selected by other expressions.	Use only if the sharding definition uses an expression for distributing data.

The following table describes the cdr change shardCollection options.

Long Form	Short Form	Description
--add	-a	Adds a database server to a sharding definition.
--drop	-d	Removes a database server from a sharding definition. If the database server that is removed uses an expression for sharding, the expression is not included in the cdr change shardCollection command.
--partitions=	-p	Changes the number of hashing partitions on each shard server when the sharding strategy is consistent hashing. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.
--replace	-r	Replaces a sharding definition when the sharding strategy is expression.

Usage

The cdr change shardCollection command can change the database servers that are part of a shard cluster. You cannot use the cdr change shardCollection command to change the sharding strategy of a shard cluster. To change the sharding strategy, you must remove the shard cluster and then re-create it with a different sharding strategy. However, you can change certain properties of the following sharding strategies:

- Consistent hash: You can change the number of hashing partitions on each shard server.
- Expression: You can change the expression that is used for distributing data.

The cdr change shardCollection command creates a new sharding definition, moves existing data to appropriate shard servers, and then removes the original sharding definition.

To delete a shard server from Enterprise Replication, remove the database server from its shard cluster by running the cdr change shardCollection command and then run the cdr delete server command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 39, 99, 196, 229.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Example: Adding a database server to a sharding definition that uses a hash algorithm

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers_1
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
```

The following command adds **g_shard_server_D** to **collection_1**:

```
cdr change shardCollection collection_1 --add g_shard_server_D
```

A new sharding definition is created for **collection_1**. All data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example

Example: Changing the number of hashing partitions on shard servers

For this example, you have a sharding definition with a consistent hashing strategy that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1
   g_shard_server_2
   g_shard_server_3
```

The following command changes the number of hashing partitions on each shard server to 4:

```
cdr change shardCollection collection_1 --partitions=4
```

The data on each shard server is redistributed into four partitions. A small amount of data might be moved between shard servers.

Example

Example: Adding multiple database servers to a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_2 db_2:joe.clients
   --type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('TX','OK')"
   g_shard_server_B "IN ('NY','NJ')"
   g_shard_server_C "IN ('AL','GA')"
   g_shard_server_D  REMAINDER
```

The following command adds the shard servers **g_shard_server_E** and **g_shard_server_F** to **collection_2**:

```
cdr change shardCollection collection_2 --add
   g_shard_server_E "IN ('CA','AZ')"
   g_shard_server_F "IN ('WA','ID')"
```

A new sharding definition is created for **collection_2**. The appropriate data from **g_shard_server_D** is redistributed to the new shard servers. The old sharding definition is deleted.

Example

Example: Removing a database server from a sharding definition

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_3 db_3:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following command removes **g_shard_server_B** from **collection_3**:

```
cdr change shardCollection collection_3 --drop g_shard_server_B
```

A new sharding definition is created for **collection_3**. The data is removed from **g_shard_server_B**. All the data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example

Example: Replacing shard servers

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_4 db_4:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following command changes the shard servers for **collection_4** to **g_shard_server_A**, **g_shard_server_C**, **g_shard_server_E**, and **g_shard_server_F**:

```
cdr change shardCollection collection_4 --replace
   g_shard_server_A
   g_shard_server_C
   g_shard_server_E
   g_shard_server_F
```

A new sharding definition is created for **collection_4**. The data is removed from **g_shard_server_B** and **g_shard_server_D**. All the data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example

Example: Replacing a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_5 db_5:joe.clients
   -t delete -k unit_number -s expression -v version
   g_shard_server_A "BETWEEN 0 and 100"
   g_shard_server_B "BETWEEN 101 and 200"
   g_shard_server_C "BETWEEN 201 and 300"
   g_shard_server_D  REMAINDER
```

The following command changes the shard servers and the expression for **collection_5**:

```
cdr change shardCollection collection_5 -r
   g_shard_server_E "BETWEEN 0 and 100"
   g_shard_server_F "BETWEEN 101 and 200"
   g_shard_server_G "BETWEEN 201 and 200"
   g_shard_server_C "BETWEEN 301 and 400"
   g_shard_server_D  REMAINDER
```

A new sharding definition is created for **collection_5**. The data is removed from **g_shard_server_A** and **g_shard_server_B**. All the data is redistributed to the appropriate database servers in the new sharding definition. The old sharding definition is deleted.

Related reference

cdr define shardCollection on page 361

cdr delete shardCollection on page 380

cdr list shardCollection on page 405

Related information

Enabling sharding for JSON or relational data on page

Changing the definition for a shard cluster on page

onstat -g shard command: Print information about the shard cache on page

Shard cluster management and monitoring on page 158

cdr check catalog

The **cdr check catalog** command compares the metadata information related to servers, replicates and replicate sets on replication servers for any inconsistency.

Syntax

```
>>-cdr check catalog--+--------------------+------------->
                               |               (1) |
                               '-| Connect Option  |-----'

>--+- --master=data_server-----+--------------------------------->

        .--------------.
       V               |
>--+---target_server-+-+----------------------------------------->
      '- --all---------'

>--+------------+------------------------------------------------>
   '- --verbose-'
```

Note:

1. See Connect Option on page 256.

Element	Purpose	Restrictions	Syntax
target_server	Name of a database server group to check.	Must be the name of an existing database server group in the sqlhosts file.	Long identifiers

The following table describes the options to **cdr check catalog**.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that master server metadata info is compared to metadata info on all servers in ER domain.
--master=	**-m**	Specifies the database server to use as the reference copy of the data.
--verbose	**-v**	Specifies that the consistency report shows all comparisons.

Usage

Use **cdr check catalog** command to compare the metadata information related to servers, replicates and replicate sets on replication servers for any inconsistency. If you include the **--verbose** option, the report lists every comparison for metadata between the master server and target servers. If the servers specified for **--connect** or **--master** options are leaf servers, parent servers are used instead.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful. If the command is not successful, one of the following error codes is returned: 1, 5, 21, 37, 48, 53, 61, 62, 99, 121, 193, 194, 195, 205. For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following command generates a consistency report comparing the metadata on the master server **g_serv1** with the metadata on the server **g_serv2**:

```
cdr check catalog --master=g_serv1 g_serv2
```

The summary consistency report shows that the metadata is consistent:

```
Verifying server definitions...
Server definitions...OK

Verifying replicate definitions...
```

```
Replicate definitions...OK

Verifying replicate participant definitions...
Replicate participant definitions...OK

Verifying replicate participants...
Replicate participants...OK

Verifying replicate set definitions...
Replicate set definitions...OK

Verifying replicate set participants...
Replicate set participants...OK
```

This report indicates that the metadata is consistent on these servers.

Related reference

cdr check replicate on page 313

cdr check replicateset on page 325

cdr check queue

Use the cdr check queue command to check the consistency of Enterprise Replication metadata, and to check the consistency of user data before running critical tasks in the Enterprise Replication domain. The command returns successfully when all of the commands that were queued when cdr check queue was run are complete.

Syntax

cdr check queue [<Connect Option> (explicit id)] **--qname=** { **cntrlq** | **sendq** | **recvq** } *queue_name* [**--wait** = { **-1** | **0** / *time* } [{ **M** | **H** | **S** }]] { *target_server* | **--all** | **--grid** = *grid_name* }

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid	Must be the name of an existing grid.	Long Identifiers on page 256
target_server	Name of a database server group on which to check the queue		

The following table describes the cdr check queue options.

Long Form	Short Form	Meaning
--all	-a	Specifies that all servers defined for the Enterprise Replication are checked
--grid	-g	Specifies the grid name
--qname	-q	Specifies the name of the queue to monitor:

Long Form	Short Form	Meaning
		• cntrlq = Control queue • sendq = Send queue • recvq = Receive queue
--wait	-w	Specifies the amount of time to wait for queues to complete before returning. Minutes are used if the time unit is not specified. -1 = Wait until all queued elements are complete 0 (default) = Do not wait for queued elements to complete; return immediately *Positive integer* = Number of hours, minutes, or seconds to wait, depending on the time unit specified: • H or h = Hours • M or m = Minutes (default) • S or s = Seconds

Usage

The cdr check queue command is used to monitor control, send, and receive queues on one or more Enterprise Replication servers and can optionally wait for queues to empty before returning.

The Enterprise Replication queues are checked at the time that the cdr check queue command runs. The time is displayed in the command output. For control and receive queues, any messages queued after the command runs are not included in the output. For the send queue, any transactions committed after the cdr check queue command runs are not included in the output.

If a leaf server name is specified with the --connect option, the system connects to the parent server to read information from the syscdr database.

Only a DBSA can run the cdr check queue command. With a non-root installation, the user who installs the server is the equivalent of the DBSA, unless the user delegates DBSA privileges to a different user.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 17, 21, 48, 62, 94, 99, 100, 196, 222.

Example

Example 1: Control queue report for all servers

The following command waits up to 10 seconds for the control queues on all replication servers to complete before generating a report.

```
cdr check queue -q cntrlq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking cntrlq queue status for server g_madras ...
cntrlq queue status for g_madras as of Mon Dec  5 12:03:19 2011:      COMPLETE
Checking cntrlq queue status for server g_delhi ...
cntrlq queue status for g_delhi as of Mon Dec  5 12:03:19 2011:       COMPLETE
Checking cntrlq queue status for server g_bombay ...
cntrlq queue status for g_bombay as of Mon Dec  5 12:03:19 2011:      COMPLETE
```

This report indicates that all of the queue items in the control queue at the time the cdr check queue command was issued are complete.

Example

Example 2: Send queue report for all servers

The following command waits up to 10 seconds for the send queues on all replication servers to complete before generating a report.

```
cdr check queue -q  sendq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking sendq queue status for server g_madras ...
Checking sendq queue status for server g_delhi ...
sendq queue status for g_delhi as of Mon Dec  5 12:04:00 2011:       COMPLETE
sendq queue status for g_madras as of Mon Dec  5 12:04:00 2011:      COMPLETE
Checking sendq queue status for server g_bombay ...
sendq queue status for g_bombay as of Mon Dec  5 12:04:01 2011:      COMPLETE
```

This report indicates that all of the queue items in the send queue at the time the cdr check queue command was issued are complete.

Example

Example 3: Send queue report that shows timeout

The following command waits up to 10 seconds for the send queues on all replication servers to complete before generating a report.

```
cdr check queue -q  sendq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking sendq queue status for server g_madras ...
sendq queue status for g_madras as of Mon Dec  5 12:04:54 2011:      COMPLETE
Checking sendq queue status for server g_delhi ...
sendq queue status for g_delhi as of Mon Dec  5 12:04:54 2011:     INCOMPLETE
```

```
 Operation timed out.
command failed -- Command timed out. (21)
```

This report indicates that the send queue for server **g_delhi** had commands that did not complete before the timeout period of 10 seconds elapsed.

Related reference

cdr delete replicate on page 374

cdr delete replicateset on page 376

cdr delete server on page 377

cdr delete grid on page 372

Related information

Monitor and troubleshooting Enterprise Replication on page 197

Deleting a Replication Server on page 166

Deleting a Replicate on page 171

Deleting a Replicate Set on page 175

Return Codes for the cdr Utility on page 262

Enterprise Replication Server administrator on page 18

cdr check replicate

The cdr check replicate command compares the data on replication servers to create a report that lists data inconsistencies and can optionally repair the inconsistent data within a replicate.

Syntax

```
cdr check replicate [ <Connect Option > (explicit id ) ] { --master=data_server (explicit id )  | --nomaster } --repl=repl_name {
target_server  | --all } [ --name=task_name ] [ --verbose ] [ --repair { [ --extratargetrows= { delete  | keep  | merge } ] |
--timestamp  [ --deletewins]  } ] [ --firetrigger=  { off  | on  | follow } ] [ --inprogress= recheck_time ] [ --background ]
[ --skipLOB ] [ --since=start_time ] [ --where=WHERE_Clause ] [ --excludeTimeSeries ] [ --ignoreHiddenTSElements ] [ --
checksum= checksum_function ]
```

Element	Purpose	Restrictions	Syntax
checksum_function	Name of the checksum function to use during consistency checking.	The function must be installed and registered on all replication servers.	Long Identifiers on page 256
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_name	Name of the replicate to check.	Must be an existing replicate.	Long Identifiers on page 256
start_time	The time from which to check updated rows.	Can have one the following formats: • *number*M = Include rows updated in the last specified number of minutes. • *number*H = Include rows updated in the last specified number of hours. • *number*D = Include rows updated in the last specified number of days. • *number*W = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh*:*mm*:*ss*" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in the `sqlhosts` file.	Long Identifiers on page 256
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers on page 256
WHERE_Clause	Clause that specifies a subset of table rows to be checked.	You cannot include a **TimeSeries** column in the WHERE clause.	WHERE clause syntax

The following table describes the options for the cdr check replicate command.

Long Form	Short Form	Meaning
--all	-a	Specifies that all servers defined for the replicate are checked.
--background	-B	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the --name= option, or the time stamp for the command if --name= is not specified.
--checksum=		Specifies the name of an existing checksum function to use during consistency checking. By default, the checksum function that is provided with the database server is run.
--deletewins	-d	Specifies that the replicate uses the delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--excludeTimeSeries		Specifies to prevent the checking of time series data.

--extratargetrows=	-e	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*):
		• delete: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers
		• keep: retain rows on the target servers
		• merge: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns.
		Note: When **cdr check replicate** is used with **-extratargetrows** (or - e) option for SEND-ONLY replicate, server displays the following warning and continue the operation:
		`WARNING: Extra row option is not applicable to Send-Only`
		` participant.`
		`'--extratargetrows' option is ignored`
		Note: When **cdr check replicate** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted:

Long Form	Short Form	Meaning
		`Error: Receive only participant '%s' can not be a master node in data synchronization task`
--firetrigger=	-T	Specifies how to handle triggers at the target servers while data is synchronizing: • off: (default) do not fire triggers at target servers during synchronization • on: always fire triggers at the target servers even if the replicate definition does not have the --firetrigger option • follow: fire triggers at target servers only if the replicate definition has the --firetrigger option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--inprogress=	-i	Specifies to spend more than the default time to recheck inconsistent rows that might be in the process of being applied on target servers. If the --inprogress= option is not set, inconsistent rows are rechecked for up to five seconds.
--master=	-m	Specifies the database server to use as the reference copy of the data. You cannot use the --master option with the --timestamp option.
--name=	-n	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--nomaster	-N	Specifies that the replicate is configured as a data consolidation system in which the multiple primary servers only send data and the single target server only receives data.
--repair	-R	Specifies that rows that are found to be inconsistent are repaired.
--repl=	-r	Specifies the name of the replicate to check.
--since=	-S	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--skipLOB	-L	Specifies that large objects are not checked.

Long Form	Short Form	Meaning
--timestamp	-t	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. The replicate must use the time stamp or delete wins conflict resolution rule. You cannot use the --master option with the --timestamp option. You cannot use this option for replicates that include **TimeSeries** columns.
--verbose	-v	Specifies that the consistency report shows specific inconsistent values.
--where=	-w	Specifies what data to check with a WHERE clause. You cannot include a **TimeSeries** column in the WHERE clause.

Usage

Use the cdr check replicate command to check the consistency of data between multiple database servers for a specific replicate. The cdr check replicate command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the --verbose option, the report lists every inconsistent row value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the --name option and specify a name for the progress report task. Then, run the cdr stats check command and specify the progress report task name.

Depending on the state of the data in your database when you run the cdr check command, the system might also run an UPDATE STATISTICS command.

If replicated transactions are active when the cdr check replicate command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the cdr check replicate command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the --inprogress option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the --inprogress option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the cdr view apply command.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Check from a specific time with the --since option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- Check a subset of the data with the --where option. For example, if you have a corrupted table fragment on a server, you can specify to check only the data in that fragment.
- Skip the checking of large objects with the --skipLOB option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.

You can run a consistency check as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the --name option so that you can monitor the check with the cdr stats check command. You can also view the command and its results in the command_history table in the **sysadmin** database. If you use the --background option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

If your replication system is configured for data consolidation and the primary servers include the S option in their participant definitions, you must include the --nomaster option.

If you include the --repair option, the cdr check replicate command repairs inconsistent rows so that they match the rows on the reference server. The cdr check replicate command uses direct synchronization as a foreground process when repairing inconsistent rows. The cdr check replicate command with the--repair option does the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs an index scan on the replication key index at both the source server and the target server to create a checksum and identify inconsistent rows.
3. Replicates inconsistent rows from the source server to the target server by doing a placeholder update of the source server, which might result in increased logging activity. Rows are not replicated to participants that include the S option in the participant definition because those participants only send data.
4. Runs a check to determine whether any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.
5. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the --inprogress option.
6. Deletes the shadow replicate.
7. Displays the consistency report.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the --repair option with the --timestamp option. If your replicates use the delete wins conflict resolution rule, also include the --deletewins option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules. The reference server in a time stamp repair is the server with the lowest replication key.

The following table describes the columns of the consistency report.

Table 25. Consistency Report Description

Column name	Description
Node	The name of the replication server.
Rows	The number of rows that are checked in the participant. If you included the --since or --where options, this number indicates the number of rows that fit the filter conditions. The number of rows that are checked with the --since option might be different on different servers, because of replication latency. Some rows might be checked on a source server to verify target server rows even if those rows on the source server did not originally fit the filter conditions.
Elements	For replicates that include **TimeSeries** columns the Elements column is shown instead of the Rows column. The Elements column shows the number of time series elements that are checked in the participant.
Extra	The number of rows on the target server that do not exist on the reference server. For the reference server, this number is always 0.
Missing	The number of rows on the reference server that do not exist on the target server. For the reference server, this number is always 0.
Mismatch	The number of rows on the target server that are not consistent with the corresponding rows on the reference server. For the reference server, this number is always 0.
Total Mismatch	For replicates that include **TimeSeries** columns the Total Mismatch column is shown instead of the Mismatch column. The Total Mismatch column shows the number of rows on the target server that are not consistent with the corresponding rows on the reference server. If the number in this column is greater than the number in the TmSr Rltd Mismatch column, the additional rows are inconsistent in a way that does not involve a **TimeSeries** column. For the reference server, this number is always 0.
TmSr Rltd Mismatch	For replicates that include **TimeSeries** columns the TmSr Rltd Mismatch column shows the number rows on the target server that do not have the same time series properties as the corresponding rows on the reference server because of time series properties. The following time series properties are checked: • Whether the **TimeSeries** column is NULL • The origin of the time series • The calendar definition • Whether the time series is regular or irregular

Table 25. Consistency Report Description (continued)

Column name	Description
	• The time series instance ID • The time series threshold For the reference server, this number is always 0.
Other Mismatch	For replicates that include **TimeSeries** columns the Other Mismatch column shows the number of elements that are in mismatched rows on the target server. For the reference server, this number is always 0.
Processed	The number of rows that are processed to correct inconsistent rows. The number of processed rows on the reference server is equal to the number of mismatched rows plus missing rows on the target servers. The number of processed rows for a target server is usually equal to the number of extra rows it has. If a row has child rows, then the number of processed rows can be greater than the number of extra rows because the child rows must be deleted as well. If the --extratargetrows option is set to keep, then extra rows are not deleted from the target and those rows are not added to the Processed column. If the --extratargetrows option is set to merge, then those rows are replicated to the reference server and are listed in the Processed column for the target server. For a time stamp repair, the time stamp or delete wins conflict resolution rule is used to determine how to process the row.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information about these error codes, see Return Codes for the cdr Utility on page 262

Example

Example 1: Summary consistency report

The following command generates a consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2
```

The summary consistency report shows that the servers are consistent:

```
                 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
g_serv2                 52         0         0         0         0

              ------   Table scan for repl1 end   ---------
```

This report indicates that the replicate is consistent on these servers.

Example

Example 2: Summary consistency report with repair

The following command generates a consistency report and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master g_serv1 --repl=repl1 --all --repair
```

The consistency report shows that the target server has extra rows:

```
                ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 67         2         2         0         2
g_serv3                 67         0         0         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

               ------   Table scan for repl1 end   ---------
```

This report indicates that **g_serv2** has two extra rows and is missing two rows. Two rows were processed on **g_serv1** to replicate the missing rows to **g_serv2**. Also, two rows were processed on **g_serv2** to delete the extra rows. Because the --extratargetrows option was not specified, the default behavior of deleting rows on the target servers that are not on the reference server occurred.

In this example, not all repaired rows were validated. Some rows might be still in the process of being applied on the target servers. Using the --inprogress option to extend the time of the validation check after the repair might prevent validation failures.

Example

Example 3: Verbose consistency report with repair

The following command generates a verbose consistency report, creates a progress report task, and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master=g_srv1 --replicate=repl1 --all --name=task1 \
 --verbose --repair
```

The verbose consistency report shows details of the repaired rows:

```

             ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Creating Shadow Repl sync_20104_1310721_1219952381
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_srv1                 424         0         0         0        11
g_srv2                 416         3        11         0         3

The repair operation completed. Validating the repaired rows ...
Validation failed for the following rows:

row missing on <g_srv2>
key:c1:424
------------------------------------------------------------------
row missing on <g_srv2>
key:c1:425
------------------------------------------------------------------
row missing on <g_srv2>
key:c1:426
------------------------------------------------------------------
marking completed on g_srv1 status 0

               ------   Table scan for repl1 end   ---------
```

This report indicates that the first check found three extra rows and 11 missing rows on the server **g_srv2**. After the repair operation and subsequent recheck, three rows were still missing on **g_srv2**. The progress report information can be accessed with the cdr stats check task1 command.

Example

Example 4: Repeating verbose consistency report without repair

The following command generates a verbose consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**, and rechecks inconsistent rows for up to 20 seconds:

```
cdr check replicate --master g_serv1 --repl=repl_1 g_serv2 --all \
--verbose --inprogress=20
```

The verbose consistency report shows details for the inconsistent rows:

```
               ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
```

```
data mismatch between g_serv1 and g_serv2
item_num:1
order_num:1011
          lname
g_serv1   Pauly
g_serv2   Pauli
--------------------------------------------------
row missing on g_serv2
item_num:1
order_num:1014
--------------------------------------------------
row missing on g_serv2
item_num:2
order_num:1014
--------------------------------------------------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         1         0

WARNING: replicate is not in sync

               ------   Table scan for repl1 end   ---------
```

This report indicates that there is one inconsistent row on **g_serv2**. The replication key for that row is the combination of the **item_num** column and the **order_num** column. The row that is inconsistent is the one that has the item number `1` and the order number `1011`. There are two rows that are missing on **g_serv2**, each identified by its replication key value.

Example

Example 5: Summary consistency report with time filter

The following command generates a summary consistency report for the data that was updated in the last five minutes:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 --since=5M
```

The consistency report shows that the servers are consistent:

```
            ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                  2         0         0         0         0
g_serv2                  2         0         0         0         0

            ------   Table scan for repl1 end   ---------
```

Only two rows were checked on each server (the Rows column) because only two rows were updated in the last five minutes.

Example

Example 6: Consistency check and repair with time filter

The following command generates a summary consistency report for the data that was updated since July 4, 2008 at 12:30:00 local time:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 \
--since="2008-07-04 12:30:00"
```

Example

Example 7: Summary consistency report and repair with data filters

The following command generates a consistency report and repairs the data where the **region** column equals `East`:

```
cdr check replicate --master=g_serv1 --repl=repl1 --repair g_serv2 \
--where="region = 'East'"
```

Example

Example 8: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for the **repl1** replicate on all replication servers:

```
cdr check replicate --repl=repl1 --all --repair --timestamp
```

The master server is not specified because the --timestamp option is used.

The consistency report shows that the three servers are not consistent:

```
           ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         4         10
g_serv2                 67         0         2         3         0
g_serv3                 67         0         5         0         4

WARNING: replicate is not in sync

          ------   Table scan for repl1 end   ---------
```

The value in the Extra column is always 0. In this example, seven rows are replicated from the **g_serv1** server to fix missing rows. The **g_serv1** server also replicated three rows to fix mismatched rows on the **g_serv2** server. The **g_serv3** server replicated four rows to resolve mismatched rows on the **g_serv1** server.

Example

Example 9: Check a replicate that includes a TimeSeries column

The following command checks the replicate named **repl2**, which includes a **TimeSeries** column:

```
cdr check replicate --repl=repl2 --master=g_3 --all
```

The following consistency report shows that the source server, **g_3**, has more elements than the target server, **g_4**:

```
           ------   Table scan for repl2 start  --------

                                                Total    TmSr Rltd
Node                  Rows     Extra   Missing  Mismatch  Mismatch Processed
---------------- --------- --------- --------- --------- --------- ---------
g_3                      1         0         0         0         0         0
g_4                      1         0         0         0         0         0

TimeSeries Column: raw_reads
                                                             Other
Node              Elements     Extra   Missing  Mismatch  Mismatch Processed
---------------- --------- --------- --------- --------- --------- ---------
g_3                      2         0         0         0         0         0
g_4                      1         0         1         0         0         0

WARNING: replicate is not in sync
              ------   Table scan for repl2 end   ---------
```

Related reference

cdr sync replicate on page 476

cdr check replicateset on page 325

cdr stats check on page 457

Related information

Interpreting the Consistency Report on page 181

Checking Consistency and Repairing Inconsistent Rows on page 180

Preparing for Role Separation (UNIX) on page 80

Indexing the ifx_replcheck Column on page 183

Increase the speed of consistency checking on page 182

Enterprise Replication Server administrator on page 18

cdr check replicateset

The cdr check replicateset command compares the data on replication servers to create a report listing data inconsistencies. Optionally you can use the command to repair the inconsistent data within a replicate.

Syntax

```
cdr check replicateset [ <Connect Option > (explicit id ) ] { --master=data_server (explicit id ) | --nomaster } --replset=repl_set
(explicit id ) { target_server | --all } [ --name=task_name ] [ --verbose ] [ --firetrigger= { off | on | follow } ] [ --
inprogress= recheck_time ] [ --background ] [ --skipLOB ] [ --since=start_time ] [ --process= number_processes ] [ --
excludeTimeSeries ] [ --ignoreHiddenTSElements ] [ --checksum= checksum_function ] [ <Repair Options> ]
```

Repair Options

```
" --repair "

"{[ --extratargetrows= { delete | keep | merge }]| --timestamp [ --deletewins] }"

"[ --enable [ --timeout = seconds ] ]"

"[ --allrepl ]"
```

Element	Purpose	Restrictions	Syntax
checksum_function	Name of the checksum function to use during consistency checking.	The function must be installed and registered on all replication servers.	Long Identifiers on page 256
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
number_processes	The number of parallel processes to use for the command.	The maximum number of processes Enterprise Replication can use is equal to the number of replicates in the replicate set.	
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_set	Name of the replicate set. Can be the name of a derived replicate set.		Long Identifiers on page 256
seconds	The number of seconds to wait for a disabled replication server to be recognized as active by other replication servers in the domain	Must be an integer value from 0 to 60.	

Element	Purpose	Restrictions	Syntax
	and how long to wait for control messages queued at peer servers to be applied at newly-enabled server.		
start_time	The time from which to check updated rows.	Can have one the following formats: • *number*M = Include rows updated in the last specified number of minutes. • *number*H = Include rows updated in the last specified number of hours. • *number*D = Include rows updated in the last specified number of days. • *number*W = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh:mm:ss*" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers on page 256

The following table describes the cdr check replicateset options.

Long Form	Short Form	Meaning
--all	-a	Specifies that all servers defined for the replicate are checked.
--allrepl	-A	Specifies that all replicates, whether they are in a replicate set or not, are repaired. You cannot use the --replset option with the --allrepl option.

Long Form	Short Form	Meaning
--background	-B	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the --name= option, or the time stamp for the command if --name= is not specified.
--checksum=		Specifies the name of an existing checksum function to use during consistency checking. By default, the checksum function that is provided with the database server is run.
--enable	-E	Enables replication on the target server if it was disabled by the cdr disable server command.
--deletewins	-d	Specifies that the replicate uses the delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--excludeTimeSeries		Specifies to prevent the checking of time series data.

--extratargetrows=	-e	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*):
		• delete: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers
		• keep: retain rows on the target servers
		• merge: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns.
		Note: When **cdr check replicate set** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation:
		`WARNING: Extra row option isn't applicable to Send-Only participant.`
		`'--extratargetrows' option is ignored`
		Note: When **cdr check replicate set** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted:

Long Form	Short Form	Meaning
		`Error: Receive only participant '%s' can not be a master node in data synchronization task`
--firetrigger=	-T	Specifies how to handle triggers at the target servers while data is synchronizing: • off: (default) do not fire triggers at target servers during synchronization • on: always fire triggers at the target servers even if the replicate definition does not have the --firetrigger option • follow: fire triggers at target servers only if the replicate definition has the --firetrigger option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--inprogress=	-i	Specifies to spend more than the default time to recheck inconsistent rows that might be in the process of being applied on target servers. If the --inprogress= option is not set, inconsistent rows are rechecked for up to five seconds.
--master=	-m	Specifies the database server to use as the reference copy of the data. You cannot use the --master option with the --timestamp option.
--name=	-n	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--nomaster	-N	Specifies that the replicate is configured as a data consolidation system in which the multiple primary servers only send data and the single target server only receives data.
--process=	-p	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--repair	-R	Specifies that rows that are found to be inconsistent are repaired.

Long Form	Short Form	Meaning
--replset	-s	Specifies the name of the replicate set to check. You cannot use the --replset option with the --allrepl option.
--skipLOB	-L	Specifies that large objects are not checked.
--since=	-S	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--timeout=	-w	Specifies the time to wait for a disabled server to be enabled.
--timestamp	-t	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. The replicate must use the time stamp or delete wins conflict resolution rule. You cannot use the --master option with the --timestamp option. You cannot use this option for replicates that include **TimeSeries** columns.
--verbose	-v	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the cdr check replicateset command to check the consistency of data between multiple database servers for a replicate set. The cdr check replicateset command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the --verbose option, the report lists every inconsistent value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the --name option and specify a name for the progress report task. Then, run the cdr stats check command and specify the progress report task name.

Depending on the state of the data in your database when you run the cdr check command, the system might also run an UPDATE STATISTICS command.

If replicated transactions are active when the cdr check replicateset command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the cdr check replicateset command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the --inprogress option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the --inprogress option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the cdr view apply command.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Skip the checking of large objects with the --skipLOB option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the --since option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.

You can significantly improve the performance of checking a replicate set by checking the member replicates in parallel. You specify the number of parallel processes with the --process option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

You can run a consistency check as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the --name option so that you can monitor the check with the cdr stats check command. You can also view the command and its results in the command_history table in the **sysadmin** database. If you use the --background option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

If your replication system is configured for data consolidation and the primary servers include the S option in their participant definitions, you must include the --nomaster option.

The cdr check replicateset command repairs inconsistent rows so that they match the rows on the reference server. During a repair of inconsistent rows, the cdr check replicateset command uses direct synchronization as a foreground process when repairing inconsistent rows. The cdr check replicateset command with the --repair option performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
3. Performs an index scan on the replication key index at both the source server and the target server to create a checksum and identify inconsistent rows.
4. Replicates inconsistent rows from the source server to the target server by doing a placeholder update of the source server, which might result in increased logging activity. Rows are not replicated to participants that include the S option in the participant definition because those participants only send data.
5. Runs a check to determine whether any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.

6. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the --inprogress option.
7. Deletes the shadow replicate.
8. Repeats steps 2 through 7 for each replicate in the replicate set.
9. Displays the consistency report.

If you have disabled a server with the cdr disable server command, you can enable it and synchronize it by using the --enable option with the --repair option. You can optionally specify a timeout period with the --timeout option.

To repair all replicates, use the --allrepl option with the --repair option.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the --repair option with the --timestamp option. If your replicates use the delete wins conflict resolution rule, also include the --deletewins option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information about these error codes, see Return Codes for the cdr Utility on page 262

Example

Example 1: Generate a consistency report

The following command uses two processes to generate a consistency report for each of the two replicates in the set in parallel for a replicate set named **replset1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicateset --master=g_serv1 --replset=replset_1 g_serv2 \
--process=2
```

The summary consistency report for the previous command might be:

```
Jan 17 2010 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
g_serv2                 52         0         0         0         0

Jan 17 2010 15:46:55 ------   Table scan for repl1 end   ---------

Jan 17 2010 15:46:46 ------   Table scan for repl2 start  --------
```

```
------   Statistics for repl2 ------
Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 48         0         0         0         0
g_serv2                 48         0         0         0         0

Jan 17 2010 15:47:05 ------   Table scan for repl2 end   ---------
```

This report indicates that the replicate set is consistent on these servers.

The consistency report for replicate sets shows a series of consistency reports for individual replicates that has the same format as the reports run with the cdr check replicate command.

Example

Example 2: Enable and synchronize a replication server

The following command enables a replication server named **g_serv2** and repairs inconsistencies by time stamp on all of its replicate sets:

```
cdr check replicateset  --repair --enable\
--timestamp --allrepl g_serv2
```

The master server is not specified because the --timestamp option is used. The replicate set name is not specified because the --allrepl option is used.

Example

Example 3: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for all replicate on all replication servers:

```
cdr check replicateset --all --repair --timestamp --allrepl
```

Related reference

cdr sync replicateset on page 481

cdr check replicate on page 313

cdr stats check on page 457

cdr disable server on page 383

Related information

Altering multiple tables in a replicate set on page 190

Checking Consistency and Repairing Inconsistent Rows on page 180

Preparing for Role Separation (UNIX) on page 80

Indexing the ifx_replcheck Column on page 183

Increase the speed of consistency checking on page 182

Repairing inconsistencies while enabling a replication server on page 184

Enterprise Replication Server administrator on page 18

cdr check sec2er

The cdr check sec2er command determines whether a high availability cluster can be converted to replication servers.

Syntax

```
cdr check sec2er [ <Connect Option > (explicit id ) ] secondary { --print }
```

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		Long Identifiers on page 256

The following table describes the cdr check sec2er option.

Long Form	Short Form	Meaning
--print	-p	Shows the commands that would be run by the cdr start sec2er command during a conversion.

Usage

You must run the cdr check sec2er command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The output of the cdr check sec2er command can show warning messages and error messages:

- Warning messages indicate possible problems for replication after the conversion. You can solve these problems after converting the cluster to replication servers.
- Error messages indicate problems preventing the conversion to replication server. You must solve all error conditions before you run the cdr start sec2er command to convert a cluster to replication servers.

Use the --print option to display the commands that are run during a conversion.

Depending on the state of the data in your database when you run the cdr check command, the system might also run an UPDATE STATISTICS command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 225.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example checks if a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** can be converted to replication servers:

```
cdr check sec2er -c priserv secserv
```

The following output of the cdr check sec2er command indicates that conversion would be successful, but that several issues should be addressed either before or after conversion:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.

Secondary conversion to ER is possible.
Errors:0000   Warnings:0003
```

The following output of the cdr check sec2er command indicates that conversion will not be successful until the CDR_QDATA_SBSPACE configuration parameter is set in the `onconfig` file on both the primary and the secondary servers:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0001   Warnings:0003
```

The following output of the cdr check sec2er command indicates that conversion will not be successful until the `sqlhosts` files on both the primary and the secondary servers are correctly configured for Enterprise Replication:

```
WARNING:CDR_SERIAL value on serv1 can cause collisions.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Dbspace  is becoming full.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
FATAL:SQLHOSTS is not set up correctly for ER.
ERROR:SQLHOSTS is not set up correctly for ER.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0004   Warnings:0003
```

The following example shows the output of the --print option, which describes the commands that will be run when the cdr start sec2er command is run on the **priserv** server. The servers are defined as replication servers. Any tables that do not have a primary key are altered to add ERKEY shadow columns. A replicate is created and started for each user table on the **priserv** server.

```
$cdr check sec2er --print serv2
Secondary conversion to ER is possible.

Errors:0000   Warnings:0000
--
```

```
--  Define ER for the first time
--
cdr define serv -c cdr1 -I cdr1

--
--  Creating Replication Key
--
dbaccess - - <<EOF
database stores_demo;
alter table 'mpruet'.classes add ERKEY;
EOF

--
--  Define the replicates
--
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=cdr1 sec2er_1_1282611664_call_type --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.call_type" \
         "select * from 'mpruet'.call_type"
cdr start repl --connect=cdr1 sec2er_1_1282611664_call_type

cdr define repl --connect=cdr1 sec2er_4_1282611664_cust_calls --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.cust_calls" \
         "select * from 'mpruet'.cust_calls"
cdr start repl --connect=cdr1 sec2er_4_1282611664_cust_calls

cdr define repl --connect=cdr1 sec2er_5_1282611664_customer --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.customer" \
         "select * from 'mpruet'.customer"
cdr start repl --connect=cdr1 sec2er_5_1282611664_customer

cdr define repl --connect=cdr1 sec2er_3_1282611664_classes --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.classes" \
         "select * from 'mpruet'.classes"
cdr start repl --connect=cdr1 sec2er_3_1282611664_classes
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related reference

cdr start sec2er on page 450

Example of creating a new replication domain by cloning on page 95

Related information

Enterprise Replication Server administrator on page 18

cdr cleanstart

The **cdr cleanstart** command starts an Enterprise Replication server with empty queues.

Syntax

```
cdr cleanstart [ <Connect Option > (explicit id ) ]
```

Usage

The **cdr cleanstart** command starts an Enterprise Replication server, but first empties replication queues of pending transactions. Use this command if synchronizing the server using the **cdr sync** command would be quicker than letting the queues process normally.

If an Enterprise Replication server was restored from a backup, but the restore did not include all log files from the replay position, or the system was not restored to the current log file, advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the **cdr cleanstart** command followed by the **cdr sync** command to synchronize the server.

You can run this command from within an SQL statement by using the SQL administration API.

Related reference

cdr start on page 443

Related information

Enterprise Replication Server administrator on page 18

cdr connect server

The **cdr connect server** command reestablishes a connection to a database server that has been disconnected with a **cdr disconnect server** command.

Syntax

```
cdr connect server [ <Connect Option > (explicit id ) ] { server_group }
```

Element	Purpose	Restrictions	Syntax
server_group	Name of database server group to resume.	The database server group must be defined for replication and be disconnected.	Long Identifiers on page 256

Usage

When you run the **cdr connect server** command, an event alarm with a class ID of 53 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Related reference

cdr define server on page 357

cdr delete server on page 377

cdr disconnect server on page 385

cdr list server on page 402

cdr modify server on page 421

cdr resume server on page 442

cdr suspend server on page 473

Enterprise Replication Event Alarms on page 221

Related information

Enterprise Replication Server administrator on page 18

cdr define grid

The cdr define grid command creates a named grid of replication servers to simply administration.

Syntax

```
cdr define grid [ <Connect Option > (explicit id ) ] grid_name { --all | server_group }
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be unique among grid names and replicate set names.	Long Identifiers on page 256
server_group	Name of a database server group to add to the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

The following table describes the cdr define grid option.

Long Form	Short Form	Meaning
--all	-a	Include all replication servers in the domain.

Usage

You must run the cdr define grid command from a replication server that is a member of an Enterprise Replication domain.

Use the --all to include all replication servers in the domain in the grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 221.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example defines a grid named **grid1** and adds two replication servers to it:

```
cdr define grid grid1 gserv1, gserv2
```

The following example defines a grid named **grid1** and adds all replication servers in the current domain to it:

```
cdr define grid grid1 --all
```

Related reference

cdr change grid on page 296

cdr list grid on page 391

cdr delete grid on page 372

Related information

Creating a grid on page 129

Enterprise Replication Server administrator on page 18

cdr define qod

The cdr define qod command defines a master server for monitoring the quality of data (QOD) for replication servers.

Syntax

```
cdr define qod [ <Connect Option > (explicit id ) ] [ --start]
```

The following table describes the cdr define qod option.

Long Form	Short Form	Meaning
--start	-s	Specifies to start quality of data monitoring.

Usage

If Connection Manager service-level agreements (SLAs) use a apply-failure or transaction-latency policy, the Connection Manager uses QOD information to decide where to route client connection requests.

Quality of data information is used for the following SLA policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.

You can start monitoring by including the --start option or by running the cdr start qod command after the cdr define qod command.

If the monitoring of data quality is already enabled, running the cdr define qod command changes the master server.

You must run the cdr define qod command from a root server.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see Return Codes for the cdr Utility on page 262.

Example

Example 1: Defining a master server

The following command defines **server_1** as the master server for quality of data monitoring:

```
cdr define qod server_1
```

After you have defined **server_1** as the master server, you must run cdr define qod on **server_1** to begin quality of data monitoring.

Example

Example 2: Defining a master server and starting quality of data monitoring

The following command connects to **server_2**, defines **server_2** as the master server, and then starts quality of data monitoring:

```
cdr define qod -c server_2 --start
```

Related reference

cdr start qod on page 444

cdr stop qod on page 467

cdr reset qod on page 437

Related information

SLA Connection Manager configuration parameter on page

Enterprise Replication Server administrator on page 18

cdr define region

The cdr define region command creates a region that contains a subset of the servers in a grid.

Syntax

```
cdr define region [ <Connect Option > (explicit id ) ] --grid=grid_name region_name server_group
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of a defined grid.	Long Identifiers on page 256
region_name	Name of the region.	Must be unique among region names, grid names, and replicate set names.	Long Identifiers on page 256
server_group	Name of a database server group to add to the region.	Must be the name of a defined grid server.	Long Identifiers on page 256

The following table describes the cdr define region option.

Long Form	Short Form	Meaning
--grid	-g	Specifies the grid that contains the servers to include in the region.

Usage

Use the cdr define region command to define a region of a grid. You can use a region name in a grid query to limit the servers on which the query is run. Regions contain servers from a single grid. You can define multiple regions for the same grid. Regions can overlap or be contained by another region. You can create an unlimited number of regions in a grid.

You cannot change a region. If you want to add or remove a grid server from a region, delete and re-create the region. Delete the region by running the cdr delete region command. Re-create the region with a different set of grid servers by running the cdr define region command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 227.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following command creates a region that contains two servers and is named **northwest**. The region is created in the grid named **mygrid1**.

```
cdr define region --grid=mygrid1 northwest server_or server_wa
```

Related reference

ifx_grid_connect() procedure on page 527

cdr define region on page 341

cdr change gridtable on page 297

cdr remaster gridtable on page 431

ifx_node_id() function on page 543

ifx_node_name() function on page 544

Related information

Defining tables for grid queries on page 147

Grid queries on page 145

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID clause on page

cdr define replicate

The cdr define replicate command defines a replicate on the specified replication servers.

Syntax

cdr define replicate [<Connect Option > (explicit id)] [<Replicate Types > (explicit id)] [<Master Replicate Options > (explicit id)] <Conflict Options > (explicit id) [<Replication to SPL Options> (explicit id)] [<Scope Options > (explicit id)] [<Frequency Options > (explicit id)] [<Special Options > (explicit id)] { *replicate* | [<Shadow Replicate Options > (explicit id)] } [*participantmodifier*]

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier on page 257
participant	Name of a participant in the replication.	The participant must exist.	Participant and participant modifier on page 257
replicate	Name of the new replicate.	The replicate name must be unique.	Long Identifiers on page 256

Usage

All servers that are specified as participants for the replicate must be online and the cdr utility must be able to connect to each participant.

To be useful, a replicate must include at least two participants. You can define a replicate that has one or no participant, but before you can use that replicate, you must use the cdr change replicate command to add more participants. You cannot start and stop replicates that have no participants.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

When you define a replicate, the replicate does not begin until you explicitly change its state to active by running the cdr start replicate command.

Important: Do not create more than one replicate definition for each row and column combination to replicate. If the participant is the same, Enterprise Replication attempts to insert duplicate values during replication.

The maximum number of replicates that you can define as participants on a particular replication server is 32767.

You can run this command from within an SQL statement by using the SQL administration API.

Replicate TypesMaster Replicate Options

The master replicate options specify whether Enterprise Replication defines the replicate as a master replicate. By default, replicates are defined as classic replicates. By default, replicates are master replicates. If you do not specify a master server, the master replicate is based on the first participant. A master replicate uses saved dictionary information about the attributes of replicated columns to verify that participants conform to the specified schema. You must specify at least one participant when you create a master replicate. All participants that are specified are verified when the cdr define replicate or cdr change replicate command is run. If any participant does not conform to the master definition, the command fails and that local participant is disabled. If a participant you specify does not contain the master replicate table, Enterprise Replication automatically creates the table on the participant, based on the master replicate dictionary information. All database servers that have master replicates must be able to establish a direct connection with the master replicate database server.

When you create a master replicate and do not include a participant modifier, the database server internally generates a participant modifier with SELECT statement that lists each column name in the table. The database server requires the individual column names to verify the schema. If the length of the SELECT statement exceeds 15 000 ASCII characters, replicate creation fails. If your column names are too long, you can create a classic replicate, which has a generated participant modifier of SELECT *.

If you do not want to verify the schema, create a classic replicate. For example, if you want to create a data consolidation system in which one server only receives data from other servers that only send data, create a classic replicate by including the --classic option.

If you do not want to verify the schema, do not create a master replicate. For example, if you want to create a data consolidation system in which one server only receives data from other servers that only send data, create a classic replicate by omitting the --master option.

Replicate Types

```
“{ --classic  <Master Replicate Options>}”
```

Master Replicate Options

```
“ --master = server ”

“[{ --empty }]”

“[ --name = { y | n }]”

“[{ --verify | [ --autocreate ]}]”
```

Element	Purpose	Restrictions	Syntax
server	Name of the database server group from which to base the master replicate definition.	The name must be the name of a database server group.	Long Identifiers on page 256

The following table describes the replicate type options.

The following table describes the master replicate options.

Long Form	Short Form	Meaning
--autocreate	-u	Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. **Note:** Tables that are created with the --autocreate option do not automatically include indexes that are not based on the replication key, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the --autocreate option require the use of these objects you must manually create those objects. You cannot use this option for replicates that include **TimeSeries** columns.
--classic		Specifies that the replicate being created is a classic replicate.
--empty	-t	Specifies that the participant on the server that is specified with the --master option is used as the basis of the master replicate, but is not added to the replicate.
--master=	-M	Specifies that the replicate being created is a master replicate. If you omit this option, the master replicate is based on the first participant.
--name=	-n	Specifies whether the master replicate has column name verification in addition to column data type verification. Valid values are: • --name=y = Default. Column names are verified to be the same on all participants. • --name=n = Column names are not verified and discrepancies can exist.
--verify	-v	Specifies that the cdr define replicate command verifies the database, tables, and column data types against the master replicate definition on all listed servers.

Replication to SPL routine

At target participant, ‘replication to SPL routine’ type replicate definition causes SPL routine to be executed instead of applying data to target table. Target participant for “replication to SPL routine? replicate definition can be configured to be same as source database, different database on the same server, or remote peer Enterprise Replication server. “Replication to SPL routine? replicate definition does not enforce the requirement to have primary key, unique index or ER key on the replicated table.

Note: Even though data is applied to stored procedure routine, target table definition must exist.

For more information see, Replication to SPL routine on page

```
|--+--splname=spl_routine_name----+-------+---------------------+--------->
   |                              |       |               .-y-. |

'--'--jsonsplname=spl_routine_name-'      '- -- cascaderepl=-+-n-+-'
```

Long Form	Meaning
--splname	Stored procedure routine name to apply data to. SPL routine must exist at all participants. Column list for SPL routine extracted from replicate participant select statement column projection list.
--jsonsplname	Stored procedure routine name to apply data to. SPL routine and table definition must exist at all participants. Input argument for SPL routine must be a JSON document. --jsonsplname option is mutually exclusive to --splname option.
--cascaderepl	Enable cascade replication. Required if replication to SPL needs to be executed for the data applied through Enterprise Replication.

--splname option stored procedure argument list:

- Optype char(1) – operation type. Values include
  - I – Insert
  - U – Update
  - D – Delete
- Soucre_id integer – Source server id. Same as group id.
- Committime integer – Transaction commit time.
- Txnid bigint – Transaction id.
- Before value column list.
- After value column list.

Note: Column list for SPL routine extracted from select statement projection list

Conflict Options

The --conflict options specify how Enterprise Replication resolves data conflicts at the database server.

Conflict Options

“ --conflict= ”

“{always | ignore | *SPL_routine* [--optimize] | timestamp [,*SPL_routine* [--optimize]] | deletewins}”

Element	Purpose	Restrictions	Syntax
SPL_routine	SPL routine for conflict resolution	The SPL routine must exist.	Long Identifiers on page 256

The following table describes the --conflict options.

Long Form	Short Form	Meaning
--conflict=	-C	Specifies the rule that is used for conflict resolution. • Use the always option if you do not want Enterprise Replication to resolve conflicts, but you do want replicated changes to be applied even if the operations are not the same on the source and target servers. Use the always-apply conflict resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent. You must use the always-apply rule if your replicate includes **TimeSeries** data types. • Use the ignore option if you do not want Enterprise Replication to resolve conflicts. • Use the timestamp option to have the row or transaction with the most recent time stamp take precedence in a conflict. • Use the deletewins option to have the row or transaction with a DELETE operation, or otherwise with the most recent time stamp, take precedence in a conflict. The delete wins conflict resolution rule prevents upserts. The action that Enterprise Replication takes depends on the scope.
--optimize	-O	Specifies that the SPL routine is optimized. An optimized SPL routine is called only when a collision is detected and the row to be replicated fails to meet one of the following two conditions:

Long Form	Short Form	Meaning
		• It is from the same database server that last updated the local row on the target table. • It has a time stamp greater than or equal to that of the local row. When this option is not present, Enterprise Replication always calls the SPL routine that is defined for the replicate when a conflict is detected.

Scope Options

The --scope options specify the scope of Enterprise Replication conflict resolution.

Scope Options

```
“ --scope= ”

“{transaction | row}”
```

The following table describes the --scope option.

Long Form	Short Form	Meaning
--scope=	-S	Specifies the scope that is used when Enterprise Replication encounters a problem with data or a conflict occurs. • --scope=row = Evaluate one row at a time and apply the replicated rows that win the conflict resolution with the target rows. • --scope=transaction = Default. Apply the entire transaction if the replicated transaction wins the conflict resolution. When you specify the scope, you can abbreviate transaction to tra.

Special Options

Special Options

```
“[{ | --ats | --ris | --floatieee | --floatcanon | --firetrigger| --fullrow={y | n}| --
ignoredel={y | n}|{ [ --erkey] [ --key column_name] | --anyUniqueKey}| --UTF8={y | n} | --
serial| --alwaysRepLOBs={y | n}}]”
```

Element	Purpose	Restrictions	Syntax
column_name	The name of a column that is included in a unique index or constraint	The column must exist.	Long Identifiers on page 256

The following table describes the special options to the cdr define replicate command.

Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • --alwaysRepLOBS=n: Default. Columns that contain unchanged large objects are not replicated. • --alwaysRepLOBS=y: Columns that contain large objects are always included in replicated rows.
--anyUniqueKey	-U	Specifies that the replication key is detected automatically from the following sources in the follow order: • A primary key that is defined on the table • ERKEY shadow columns that are included in the table • Any unique key or unique constraint that is defined on the table The replicate must be a strictly mastered replicate. If the table includes ERKEY shadow columns, those columns are included in the participant definition only if the table does not have a primary key.
--ats	-A	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database.
--erkey	-K	Adds the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, to the participant definition, if the table includes the ERKEY shadow columns. An index that is created on the ERKEY shadow columns is used as the replication key, unless the --key option is included.
--firetrigger	-T	Specifies that the rows that the replicate inserts fire triggers at the destination.
--floatieee	-I	Transfers replicated floating-point numbers in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Use this option for all new replicate definitions.

Long Form	Short Form	Meaning
--floatcanon	-F	Transfers replicated floating-point numbers in machine-independent decimal representation. This format is portable, but can lose accuracy. This format is provided for compatibility with earlier versions only; use --floatieee for all new replicate definitions.
--fullrow=	-f	Specifies whether to replicate full rows or only the changed columns: • --fullrow=y = Default. Indicates to replicate the full row and to enable upserts. If you also specify deletewins as the conflict resolution rule, upserts are disabled. • --fullrow=n = Indicates to replicate only changed columns and disable upserts.
--ignoredel=	-D	Specifies whether to retain deleted rows on other nodes: • --ignoredel=y = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify deletewins as the conflict resolution rule. • --ignoredel=n = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--key=	-k	Specifies the columns that are included in an existing unique index or unique constraint to use as the replication key. All the columns that are included in the unique index or constraint must be listed, in the same order as the columns are listed in the index or constraint definition. The replicate must be a strictly mastered replicate. The unique index or constraint that the --key option specifies is used as the replication key even if the table has an existing primary key or ERKEY columns.
--ris	-R	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems.
--serial	-s	Specifies that replicated transactions for the replicate are applied serially instead of in parallel.
--UTF8=	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets.

Long Form	Short Form	Meaning
		• --UTF8=y Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types, such as user-defined types or DataBlade® data. No attempt is made to convert character data that is contained within opaque data types. You cannot use --UTF8=y for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade® module data types. • -UTF8=n Indicates that code set conversion is ignored.

Shadow Replicate Options

A shadow replicate is a copy of an existing, or primary, replicate. You must create a shadow replicate to manually remaster of a replicate that is defined with the -n option. After you create the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate by running the cdr swap shadow command.

Shadow Replicate Options

“ [`--mirrors`*primary_replicate shadow_replicate*] ”

Element	Purpose	Restrictions	Syntax
primary_replicate	Name of the replicate on which to base the shadow replicate.	The replicate must exist. The replicate name must be unique.	Long Identifiers on page 256
shadow_replicate	Name of the shadow replicate to create.	The replicate name must be unique.	Long Identifiers on page 256

The following table describes the shadow replicate option to cdr define replicate.

Long Form	Short Form	Meaning
--mirrors	-m	Specifies that the replicate created is a shadow replicate based on an existing primary replicate.

Example

Example 1: Define a replicate with two participants

The following example defines a replicate with two participants:

```
cdr define repl --conflict=timestamp,sp1 \
--scope=tran --ats --fullrow=n --floatieee newrepl \
"
db1@iowa:antonio.table1"
 "
select * from table1"
 \
"
db2@utah:carlo.table2"
 "
select * from table2"
```

Line 1 of the example specifies a primary conflict resolution rule of timestamp. If the primary rule fails, the SPL routine sp1 is run to resolve the conflict. Because no database server is specified here (or on any later line), the command connects to the database server named in the **INFORMIXSERVER** environment variable.

Line 2 specifies that the replicate has a transaction scope for conflict resolution scope and enables aborted transaction spooling. Enterprise Replication replicates only the rows that changed and uses the IEEE floating point format to send floating-point numbers across dissimilar platforms. The final item specifies the name of the replicate, **newrepl**.

Line 3 defines the first participant, **"db1@iowa:antonio.table1"**, with the SELECT statement **"select * from table1"**.

Line 4 defines a second participant, "**db2@utah:carlo.table2**", with the SELECT statement "**select * from table2**".

Example

Example 2: Define a replicate with a frequency of every five hours

This example is the same as the preceding example with the following exceptions:

- Line 1 instructs Enterprise Replication to use the global catalog on database server **ohio**.
- Line 2 specifies that the data is replicated every five hours.

```
cdr def repl -c ohio -C timestamp,sp1 \
-S tran -A -e 5:00 -I newrepl \
"db1@iowa:antonio.table1" "select * from table1" \
"db2@utah:carlo.table2" "select * from table2"
```

Example

Example 3: Define a master replicate

The following example defines a master replicate:

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl \
"db1@iowa:antonio.table1" "select * from table1"
```

Line 1 instructs Enterprise Replication to create a master replicate based on the replicate information from the database server **iowa**. Line 2 specifies the delete wins conflict resolution rule, a transaction scope, and that the name of the replicate is **newrepl**. Line 3 specifies the table and columns included in the master replicate.

Example

Example 4: Define a master replicate and create a table on a participant

This example is the same as the previous example except that it specifies a second participant in Line 4. The second participant (**utah**) does not have the table **table1** specified in its participant and modifier syntax. The -u option specifies to create the table **table1** on the **utah** server.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl -u\
"db1@iowa:antonio.table1" "select * from table1 \
"db2@utah:carlo.table1" "select * from table1"
```

Example

Example 5: Define a replicate with the ERKEY shadow columns

This example defines a master replicate similar to the one in example 3, and includes the ERKEY shadow columns for the replication key.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl --erkey\
"db1@iowa:antonio.table1" "select * from table1"
```

Example

Example 6: Define a data consolidation system

This example defines a replicate for a data consolidation system in which one target server receives replicated data from four primary servers.

```
cdr def repl -c london \
sales -C always\
"db0@london:user.world_sales" "select * from world_sales"\
"S db1@rome:user1.sales_rome" "select * from sales_rome"\
"S db2@tokyo:user2.sales_tokyo" "select * from sales_tokyo"\
"S db3@perth:user3.sales_perth" "select * from sales_perth"\
"S db4@ny:user4.sales_ny" "select * from sales_ny"\
```

The S options in the participant definitions indicate that the **rome**, **tokyo**, **perth**, and **ny** servers can only send replicated data to the **london** server.

Related reference

cdr change replicate on page 299

cdr change replicateset on page 302

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

cdr swap shadow on page 475

cdr define template on page 367

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

Replicate only changed columns on page 106

Participant and participant modifier on page 257

Related information

Enabling ATS and RIS File Generation on page 200

Specifying Conflict Resolution Rules and Scope on page 104

Replicate types on page 101

Conflict Resolution on page 40

Conflict Resolution Scope on page 50

Enabling Triggers on page 107

Using the IEEE Floating Point or Canonical Format on page 107

Failed Transaction (ATS and RIS) Files on page 199

Frequency Options on page 287

Defining Replication Servers on page 93

Setting Up Failed Transaction Logging on page 105

Preparing for Role Separation (UNIX) on page 80

Preparing tables without primary keys on page 79

Replication of TimeSeries data types on page 30

Enterprise Replication Server administrator on page 18

cdr define replicateset

The cdr define replicateset command defines a replicate set on all the servers that are included as participants in the replicates. A replicate set is a collection of several replicates to be managed together.

Important: Enterprise Replication supports replicate sets for HCL Informix®, Version 9.3 and later only. You cannot define or modify replicate sets to include replicates with participants that are version 9.2 and earlier. In addition, replicate sets are different from and are incompatible with replicate groups (in version 9.2 and earlier).

Syntax

```
cdr define replicateset [ <Connect Option > (explicit id ) ] [ <Frequency Options > (explicit id ) ] [ --exclusive ] { repl_set [
replicate ] | --needRemaster=original_setderived_set }
```

Element	Purpose	Restrictions	Syntax
derived_set	Name of a temporary replicate set to create that contains only replicates that must be remastered.	The name must be unique and cannot be the same as a replicate name.	Long Identifiers on page 256
original_set	Name of an existing replicate set that contains replicates that must be remastered.	The replicate set must exist.	Long Identifiers on page 256
repl_set	Name of replicate set to create.	The name must be unique and cannot be the same as a replicate name.	Long Identifiers on page 256
replicate	Name of a replicate to be included in the replicate set.	The replicate must exist.	Long Identifiers on page 256

The following table describes the options to the cdr define replicateset command.

Long Form	Short Form	Meaning
--exclusive	-X	Creates an exclusive replicate set. Replicates that belong to this replicate set cannot belong to any other replicate sets.
--needRemaster=	-n	Creates a derived replicate set that contains replicates that have schema changes and must be remastered, and any classic replicates. All classic replicates are converted to master replicates regardless of whether they have schema changes.

Usage

All servers that are specified as participants for all the specified replicates must be online and the cdr utility must be able to connect to each participant.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

Any valid replicate can be defined as part of a replicate set. A replicate can belong to more than one non-exclusive replicate set, but to only one exclusive replicate set.

When you create an exclusive replicate set, the state is initially set to active.

To create an exclusive replicate set and make it active

1. Create an empty replicate set.
2. Stop the replicate set.
3. Add replicates to the replicate set.
4. Set the state of the replicate set to active by running cdr start replicateset.

Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state. You cannot change whether a replicate set is exclusive or not.

If you change the schema of multiple replicated tables for replicates that belong to the same replicate set, you can create a derived replicate set so that you can remaster all the replicates with one command. Use the --needRemaster option to specify the existing replicate set and the name of the derived replicate set. Then run the cdr remaster replicateset command.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Example: Define a non-exclusive replicate set

The following command connects to the default server and defines the non-exclusive replicate set accounts_set with replicates **repl1**, **repl2**, and **repl3**:

```
cdr def replset accounts_set repl1 repl2 repl3
```

Example

Example: Define an exclusive replicate set

The following command connects to the server **olive** and defines the exclusive replicate set **market_set** with replicates **basil** and **thyme**:

```
cdr def replset --connect=olive --exclusive market_set basil thyme
```

Example

Example: Define a derived replicate set

The following command defines a derived replicate set named **derived_accounts** that is based on the replicate set **accounts_set**:

```
cdr define replicateset --needRemaster=accounts_set derived_accounts
```

Related reference

cdr change replicateset on page 302

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

Related information

Altering multiple tables in a replicate set on page 190

Exclusive Replicate Sets on page 118

Define replicate sets on page 117

Frequency Options on page 287

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr define server

The cdr define server command defines a replication server in an Enterprise Replication domain. You can add a replication server to an existing domain or create a new domain.

Syntax

cdr define server [<Connect Option > (explicit id)] [<Dynamic Options>] [**--sync=***sync_server* [{ **--nonroot** | **--leaf** }]] **--init** *server_group*

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to add to an Enterprise Replication domain.	Must be the name of an existing database server group in the `sqlhosts` information.	

Element	Purpose	Restrictions	Syntax
sync_server	Name of a replication server that is a member of the domain into which you are adding a server.	Must be an existing replication server. The server must be online.	Long Identifiers on page 256

The following table describes the options to cdr define server.

Long Form	Short Form	Meaning
--init	-I	Adds *server_group* to the replication domain.
--leaf	-L	Defines the server as a leaf server in an existing domain. The server that is specified by the --sync option becomes the parent of the leaf server.
--nonroot	-N	Defines the server as a nonroot server in an existing domain. The server that is specified by the --sync option becomes the parent of the nonroot server.
--sync=	-S	Adds a server to the existing domain of which the *sync_server* is a member. Uses the global catalog on *sync_server* as the template for the global catalog on the new replication server, *server_group*. For Hierarchical Routing topologies, Enterprise Replication also uses the *sync_server* as the parent of the new server in the current topology.

Dynamic Options

Use the dynamic options to modify the default behavior of cdr define server. You can change these options with the cdr modify server command while replication is active.

Options

“[**--ats=***ats_dir*][**--ris=***ris_dir*][**--atsrisformat=**{ **text** | **xml** | **both** }][**--idle=***timeout*]”

Element	Purpose	Restrictions	Syntax
ats_dir	Name of the directory for Aborted Transaction Spooling files. The default is `/tmp`.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of `/dev/null` (UNIX™) or `NUL` (Windows™) prevents ATS file generation.	Follows naming conventions on your operating system

Element	Purpose	Restrictions	Syntax
ris_dir	Name of the directory for Row Information Spooling files. The default is `/tmp`.	Must be a full path name. The path for the directory can be no longer than 256 characters. A value of `/dev/null` (UNIX™) or `NUL` (Windows™) prevents RIS file generation.	Follows naming conventions on your operating system
timeout	Idle timeout for this replication server.	Default value of 0 indicates no timeout. Must be an integer number of minutes. The maximum value is 32767.	Integer

The following table describes the options to cdr define server that you can change with the cdr modify server command while replication is active.

Long Form	Short Form	Meaning
--ats=	-A	Specifies the directory to store aborted transaction spooling files for replicate transactions that fail to be applied.
--atsrisformat=	-X	Specifies the format of ATS and RIS files: • text indicates that ATS and RIS files are generated in standard text format. • xml indicates that ATS and RIS files are generated in XML format. • both indicates that ATS and RIS files are generated in both standard text format and XML format. If you omit the --atsrisformat= option, ATS and RIS files are created in text format.
--ris=	-R	Specifies the directory to store row information spooling files for replicate row data that fails conflict resolution or encounters replication-order problems.
--idle=	-i	The default value is `0`. Set the number of minutes after which an inactive connection is closed after *timeout* minutes. If timeout is `0`, the connection does not time out.

Usage

Run the cdr define server command on the database server that you want to define as a replication server. To create the replication server in an existing domain, specify a synchronization server that belongs to that domain with the --sync= option. To create a replication server in a new domain, omit the --sync= option. The cdr define server command creates the Enterprise Replication global catalog on the specified server.

If the CDR_QDATA_SBSPACE and CDR_DBSPACE configuration parameters are not set and the database server has a storage pool with sufficient space, the cdr define server command automatically performs the following tasks:

- Creates an sbspace and a dbspace from chunks from the storage pool. The spaces have the same characteristics and storage-pool behavior as other spaces created from the storage pool. The storage pool must have at least 500 MB of free space for the sbspace and 200 MB of free space for the dbspace. These spaces must be composed of chunks of size 100 MB or greater.
- Sets the values of the CDR_QDATA_SBSPACE and CDR_DBSPACE configuration parameters to the space names both in memory and in the `onconfig` file.
- Shows the names of the spaces that are created.

You can run this command from within an SQL statement by using the SQL administration API.

The maximum number of replication servers that you can define is 32767.

Example

Examples

The following example defines the first database server in a replication domain. The command specifies the following actions:

- Connect to the database server **stan**.
- Initialize Enterprise Replication.
- Set the `/cdr/ats` directory for generated ATS files.
- Set the `/cdr/ris` directory for generated RIS files.
- Set the format of ATS and RIS files to text.

```
cdr define server --connect=stan \
--ats=/cdr/ats --ris=/cdr/ris \
--atsrisformat=text --init g_stan
```

The following example adds a database server to the replication domain that was created in the previous example. The command specifies the following actions:

- Connect to the database server **oliver**.
- Initialize Enterprise Replication.
- Synchronize the catalogs on database server **oliver** with the catalogs on database server **stan**.
- Set the `/cdr/ats` directory for generated ATS files.

- Specify that RIS files are not generated.
- Set the file format of ATS files to XML.

```
cdr define server -c oliver \
-A /cdr/ats -R /dev/null -X xml \
-S g_stan -I g_oliver
```

Related reference

cdr connect server on page 337

cdr delete server on page 377

cdr disconnect server on page 385

cdr list server on page 402

cdr modify server on page 421

cdr resume server on page 442

cdr suspend server on page 473

cdr realize template on page 423

Related information

Enabling ATS and RIS File Generation on page 200

Disabling ATS and RIS File Generation on page 212

Monitor and troubleshooting Enterprise Replication on page 197

Defining Replication Servers on page 93

Customizing the Replication Server Definition on page 99

Enterprise Replication Server administrator on page 18

cdr define shardCollection

The cdr define shardCollection command creates a sharding definition for distributing a table or collection across multiple shard servers.

Syntax

```
>>-cdr define shardCollection----------------------------------->

>--definition_name--database--:--user--.--+-collection-+--------->
                                          '-table------'

                                           .-delete--------.
>--+------------------------+-- --type--=--+-keep----------+---->
   |                    (1) |              '-informational-'
   |                        |              '-informational_noer-'
   '-| Connect Option |-----'
```

```
>-- --key--=--+-column-----------+------------------------------>
              '-"--expression--"-'

>--+------------------------------+----------------------------->
   '- --versionCol--=--+-field--+-'
                       '-column-'

                                 .----------------------------------.
                                 V                                  |
>-- --strategy--=--+-expression----ER_group--"--WHERE_expression--"-+--ER_group--REMAINDER-+-><
                   |         .----------.                                                  |
                   |         V          |                                                  |
                   +- chash----ER_group-+--+------------------------------+----------------+
                   |                       '- --partitions--=--partitions-'             |
                   |        .----------.                                                   |
                   |        V          |                                                   |
                   '- hash----ER_group-+---------------------------------------------------'
```

Element	Description	Restrictions
collection	The name of the collection that is distributed across database servers.	Must be the name of an existing collection.
column	The name of a table column.	Must be the name of an existing column.
database	The name of the database that contains the table or collection that is distributed across database servers.	Must be the name of an existing database.
definition_name	The name of the sharding definition that is created.	
ER_group	The ER-group name of a database server that receives sharded data.	Must be the ER-group name of an existing database server.
expression	The WHERE-clause expression that is used to select rows or documents by shard key value.	
field	The name of a collection field.	Must be the name of an existing field.
REMAINDER	Specifies the database server that receives rows or documents with shard key value that is not selected by the other expressions.	
partitions	The number of hashing partitions to create on each shard server.	Must be a positive integer.
table	The name of the table that is distributed across database servers.	Must be the name of an existing table.

Element	Description	Restrictions
user	The owner of the table or collection that is distributed across database servers.	Must be the name of an existing user.

The following table describes the cdr define shardCollection parameters.

Long Form	Short Form	Description
--key=	-k	Defines the shard key on all database servers. Possible values are: • A column name • An expression All database servers in a shard cluster must use the same column or expression as the shard key.
--partitions=	-p	Specifies the number of hashing partitions to create on each shard server when the sharding strategy is consistent hashing. Default is 3. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.
--strategy=	-s	Specifies the method for determining which database server an inserted row or document is distributed to. Possible values are: • **expression**: The expression that is defined in the server statement is used. • **chash**: A consistent hash algorithm is used. When you add or remove a shard server, the consistent hashing algorithm redistributes a fraction of the data. • **hash**: A hash algorithm is used. When you add or remove a shard server, the hashing algorithm redistributes all the data.
--type=	-t	Specifies action on the shard server where a row or document was inserted: • **delete** (default): The row or document is deleted from the source shard server after it is replicated to the target shard server. If you do not set `--versionCol=`*column*, changes made to rows and documents can be lost during the replication process.

Long Form	Short Form	Description
		- **keep**: The row or document is not deleted on the source shard server after the row or document is replicated to the source shard serverso that two copies of the data exist in the shard cluster. - **informational**: Data is not replicated. You can run sharded queries but the data is not sharded during loading. You must load the data on the appropriate shard server according to the sharding definition. - **informational_noer**: You can use with shard edge servers without having to define Enterprise Replication at edge servers.
--versionCol=	-v	When `--type=delete` is specified in the sharding definition, Enterprise Replication must verify that a source row or document was not updated before it can delete the row or document on the shard server. Possible values are: - A table column name - A field name If `--type=delete` is set in the sharding definition, but `--versionCol=`*column* is not, changes made to rows and documents can be lost during the replication process. This parameter is required if any rows have out-of-row data, such as data stored in smart large object, or if collections have BSON documents that have sizes larger than 4 KB.

Usage

Use the cdr define shardCollection command to create a sharding definition for distributing a table or document across multiple shard servers. The replicates that are created as part of the cdr define shard command are mastered and use always apply and row scope. You cannot specify that triggers fire.

Multiple sharding definitions are not allowed on the same table or collection.

You cannot manually define an Enterprise Replication replicate for a table that is sharded.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 3, 18, 39, 52, 83, 99, 125, 196, 215, 229.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Example: Creating a sharding definition that uses a consistent hash algorithm

The following example creates a sharding definition that is named **collection_1**. Rows that are inserted on any of the shard servers are distributed, based on a consistent hash algorithm, to the appropriate shard server. Enterprise Replication must verify that a replicated row or document was not updated before the row or document can be deleted on the source server. The **b** column in the **customers** table that is owned by user **john** is the shard key. Each shard server has three hashing partitions.

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

The partition range for each shard server is calculated based on the server group name. The data is distributed according to the following sharding definition:

```
g_shard_server_1     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
g_shard_server_2     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
g_shard_server_3     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950)
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 40
```

Example

Example: Creating a sharding definition that uses a hash algorithm

The following example creates a sharding definition that is named **collection_1**. Rows that are inserted on any of the shard servers are distributed, based on a hash algorithm, to the appropriate shard server. Enterprise Replication must verify that a replicated row or document was not updated before the row or document can be deleted on the source server. The **state** column in the **customers** table that is owned by user **john** is the shard key.

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A g_shard_server_B g_shard_server_C g_shard_server_D
```

Example

Example: Creating a sharding definition that uses an IN expression

The following example creates a sharding definition that is named **collection_2**. The **state** column in the **clients** table that is owned by user **joe** is the shard key. Rows that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server. Replication acknowledgment must verify that a replicated row or document was not updated before the row or document can be deleted on the source shard server.

```
cdr define shardCollection collection_2 db_2:joe.clients
--type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('TX','OK')"
   g_shard_server_B "IN ('NY','NJ')"
   g_shard_server_C "IN ('AL','GA')"
   g_shard_server_D  REMAINDER
```

In the previous example:

- Inserted rows that have a value of `AL` in the **state** column are sent to **g_shard_server_C**.
- Inserted rows that have a value of `NJ` in the **state** column are sent to **g_shard_server_B**.
- Inserted rows that have a value of `CA` in the **state** column are sent to **g_shard_server_D**.

Example

Example: Creating a sharding definition that uses a BETWEEN expression

The following example creates a definition that is named **collection_3**. The **age** column in the **users** table that is owned by user **charles** is the shard key. Rows that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server. Replication acknowledgment must verify that a replicated row or document was not updated before the row or document can be deleted on the source shard server.

```
cdr define shardCollection collection_3 db_3:charles.users
--type=delete --key=age --strategy=expression --versionCol=version
   g_shard_server_A "BETWEEN 0 and 20"
   g_shard_server_B "BETWEEN 21 and 62"
   g_shard_server_C "BETWEEN 63 and 100"
   g_shard_server_D  REMAINDER
```

In the previous example:

- Inserted rows that have a value of `35` in the **age** column are sent to **g_shard_server_B**.
- Inserted rows that have a value of `102` in the **age** column are sent to **g_shard_server_D**.
- Inserted rows that have a value of `15` in the **age** column are sent to **g_shard_server_A**.

Example

Example: Creating a sharding definition that defines a shard key by function

The following example creates a sharding definition that is named **collection_4**. The **COLOR** shard key in the **cars** collection that was owned by user **mike** is the shard key. Documents that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server.

```
cdr define shardCollection collection_4 db_4:mike.cars
   -t delete -k "bson_value_lvarchar(data,'COLOR')" -s expression -v version
   g_shard_server_E "IN ('blue','green')"
   g_shard_server_F "IN ('black','white')"
   g_shard_server_G "IN ('brown','gray')"
   g_shard_server_H "IN ('red','yellow')"
   g_shard_server_I  REMAINDER
```

In the previous example:

- Inserted documents that have a value of `yellow` in the **COLOR** key are sent to **g_shard_server_H**.
- Inserted documents that have a value of `blue` in the **COLOR** key are sent to **g_shard_server_E**.
- Inserted documents that have a value of `pink` in the **COLOR** key are sent to **g_shard_server_I**.

Related reference

cdr change shardCollection on page 304

cdr delete shardCollection on page 380

cdr list shardCollection on page 405

Related information

Creating a shard cluster on page 153

Enabling sharding for JSON or relational data on page

Creating a shard cluster by running the addShard command in the MongoDB shell on page

Creating a shard cluster by running the addShard command through db.runCommand in the MongoDB shell on page

Creating a shard-cluster definition that uses a hash algorithm for distributing data across database servers on page

onstat -g shard command: Print information about the shard cache on page

cdr define template

The cdr define template command creates a template for replicates and a replicate set.

Because templates define replicates, many of the syntax options for the cdr define template command are the same as for the cdr define replicate command.

Syntax

```
cdr define templatetemplate [ <Connect Option > (explicit id ) ] <Conflict Options > (explicit id ) [ <Scope Options > (explicit
id ) ][ <Special Options > (explicit id ) ] --master=server_group [ --exclusive ] --database=database { table | --all | --
file=filename }
```

Element	Purpose	Restrictions	Syntax
template	Name of the template to create.	The template name must be unique and cannot be the same as a replicate or replicate set name.	Long Identifiers on page 256
database	Name of the database that is used to define the template attributes.	The database server must be registered with Enterprise Replication.	Long Identifiers on page 256
table	Name of the table to be included in the template.	The table must be an actual table. It cannot be a synonym or a view. For	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
		ANSI databases, you must specify *owner.tablename*.	
filename	The directory and file name of the file that contains a list of tables to be included in the template.	Must be a full path name and file name. The path and file name can be no longer than 256 bytes. Within the file, the table names can be separated by a space or placed on different lines.	Follows naming conventions on your operating system.
server_group	Name of a database server group to declare for Enterprise Replication.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

The following table describes the options to cdr define template.

Long Form	Short Form	Meaning
--all	-a	Specifies that all tables in the database are included in the template.
--database=	-d	Specifies which database the template is based on. If no tables or table list file name are listed after this option, then all tables in the database are included in the template.
--exclusive	-X	Creates an exclusive replicate set. The state of the replicate set is inactive until you apply the template. This option is required if you have referential integrity constraints on a table.
--file=	-f	Specifies the path and file name of a file that lists the tables to be included in the template. The file must contain only table names, either separated by spaces or each on its own line.
--master=	-M	Specifies the server that contains the database to be used as the basis of the template. If this option is not specified, then the server that is specified in the connect option is used.

Special Options

Special Options

```
“[{ | --ats | --ris | --floatieee | --floatcanon | --firetrigger| --fullrow={y | n}| --
ignoredel={y | n}|{ --anyUniqueKey}| --UTF8={y | n}| --alwaysRepLOBs={y | n}}]”
```

The following table describes the special options to the cdr define template command.

Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • --alwaysRepLOBS=n: Default. Columns that contain unchanged large objects are not replicated. • --alwaysRepLOBS=y: Columns that contain large objects are always included in replicated rows.
--anyUniqueKey	-U	Specifies that the replication key is detected automatically from the following sources in the follow order: • A primary key that is defined on the table • ERKEY shadow columns that are included in the table • Any unique key or unique constraint that is defined on the table The replicate must be a strictly mastered replicate. If the table includes ERKEY shadow columns, those columns are included in the participant definition only if the table does not have a primary key.
--ats	-A	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database.
--firetrigger	-T	Specifies that the rows that the replicate inserts fire triggers at the destination.
--floatieee	-I	Transfers replicated floating-point numbers in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Use this option for all new replicate definitions.
--floatcanon	-F	Transfers replicated floating-point numbers in machine-independent decimal representation. This format is portable, but can lose accuracy. This format is provided for compatibility with earlier versions only; use --floatieee for all new replicate definitions.
--fullrow=	-f	Specifies whether to replicate full rows or only the changed columns:

Long Form	Short Form	Meaning
		• --fullrow=y = Default. Indicates to replicate the full row and to enable upserts. If you also specify deletewins as the conflict resolution rule, upserts are disabled. • --fullrow=n = Indicates to replicate only changed columns and disable upserts.
--ignoredel=	-D	Specifies whether to retain deleted rows on other nodes: • --ignoredel=y = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify deletewins as the conflict resolution rule. • --ignoredel=n = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris	-R	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems.
--UTF8=	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets. • --UTF8=y Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types, such as user-defined types or DataBlade® data. No attempt is made to convert character data that is contained within opaque data types. You cannot use --UTF8=y for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade® module data types. • -UTF8=n Indicates that code set conversion is ignored.

Usage

A template consists of schema information about a database, a group of tables, column attributes, and the replication keys that identify rows. A template defines a group of master replicates and a replicate set. Templates are an alternative to using the cdr define replicate and cdr start replicate commands for each table and manually combining the replicates into a replicate set by using the cdr define replicateset command.

The replicate set can be exclusive or non-exclusive. Specify that the replicate set is exclusive if you have referential constraints that are placed on the replicated columns. If you create an exclusive replicate set using a template, you do not stop the replicate set to add replicates. The cdr define template command performs this task automatically.

If your tables include the ERKEY shadow columns, they are automatically added to replicate definition when you define a template. The --erkey option is not needed with the cdr define template command.

You cannot specify an SPL routine for conflict resolution when you define a template.

After you define a template by running the cdr define template command, use the cdr realize template command to apply the template to your Enterprise Replication database servers.

You cannot update a template. To modify a template, you must delete it and then re-create it with the cdr define template command.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example illustrates the cdr define template command:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran \
--master=chicago\
--database=new_cars table1 table2 table3
```

Line 1 of the example specifies a template name of **tem1** and that the connection is made to the server **detroit**. Line 2 specifies a conflict-resolution rule of **timestamp** and a transaction scope for conflict resolution. Line 3 specifies that the master replicate information is obtained from the server **chicago**. Line 4 specifies to use the **new_cars** database on the **chicago** server and to include only the tables **table1**, **table2**, and **table3**.

The next example is the same as the first except that it has additional options and uses a file instead of a list of tables:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran --master=chicago\
--ignoredel=y\
--database=new_cars --file=tabfile.txt
```

Line 3 indicates that delete operations are not replicated. Retaining deleted rows on target servers is useful for consolidation models.

Line 4 specifies a file name for a file that contains a list of tables to include in the template. The `tabfile.txt` file has the following contents:

```
table1
table2
table3
table4
```

Related reference

cdr list template on page 410

cdr realize template on page 423

cdr delete template on page 381

cdr define replicate on page 342

Participant and participant modifier on page 257

Related information

Exclusive Replicate Sets on page 118

Replicate types on page 101

Frequency Options on page 287

Preparing for Role Separation (UNIX) on page 80

Preparing tables without primary keys on page 79

Creating replicated tables through a grid on page 136

Enterprise Replication Server administrator on page 18

cdr delete grid

The cdr delete grid command deletes the specified grid.

Syntax

```
cdr delete grid[  <Connect Option > (explicit id ) ] grid_name
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256

Usage

Use the cdr enable grid command to delete an existing grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 222.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example deletes the grid named **grid1**:

```
cdr delete grid grid1
```

Related reference

ifx_grid_connect() procedure on page 527

cdr define grid on page 338

cdr check queue on page 310

Related information

Enterprise Replication Server administrator on page 18

cdr delete region

The cdr delete region command deletes a region from a grid.

Syntax

```
cdr delete region [ <Connect Option > (explicit id ) ] --region=region_name
```

Element	Purpose	Restrictions	Syntax
region_name	Name of the region.	Must be the name of a defined region.	Long Identifiers on page 256

The following table describes the cdr delete region option.

Long Form	Short Form	Meaning
--region	-r	Specifies the region to delete.

Usage

Use the cdr delete region command to remove a region from a grid. Delete a region that is no longer valid, for example, if you remove a grid server that is included in the region from the grid. If you need to change the list of grid servers in a region, you delete the region and then re-create it with the new server list.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 227.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following command deletes the region that is named **northwest**:

```
cdr delete region --region=northwest
```

Related reference

ifx_grid_connect() procedure on page 527

cdr delete region on page 373

cdr change gridtable on page 297

cdr remaster gridtable on page 431

ifx_node_id() function on page 543

ifx_node_name() function on page 544

Related information

Defining tables for grid queries on page 147

Grid queries on page 145

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID clause on page

cdr delete replicate

The cdr delete replicate command deletes a replicate.

Syntax

`cdr delete replicate` [`<Connect Option>` (explicit id)] *replicate_name*

Element	Purpose	Restrictions	Syntax
replicate_name	Name of the replicate to delete.		Long Identifiers on page 256

Usage

The cdr delete replicate command deletes the replicate *repl_name* from the global catalog. All replication data for the replicate is purged from the send queue at all participating database servers. You can run this command from within an SQL statement by using the SQL administration API.

Important: Avoid deleting a replicate and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later.

Important: If you are creating a replication server to replace the one you deleted, use the cdr check queue --qname=ctrlq command to make sure that the delete operation propagated to all the servers.

When you run the cdr delete replicate command, an event alarm with a class ID of 68 is generated, if that event alarm is enabled.

Example

Example 1: Deleting a single replicate

The following command connects to the default database server specified by the **INFORMIXSERVER** environment variable and deletes the replicate **reynolds**:

```
cdr delete replicate reynolds
```

Example

Example 2: Deleting multiple replicates

The following command connects to database server **hoek** and deletes the replicates **reynolds** and **stimpson**:

```
cdr del rep -c hoek reynolds stimpson
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

cdr check queue on page 310

Enterprise Replication Event Alarms on page 221

Related information

Deleting a Replicate on page 171

Enterprise Replication Server administrator on page 18

cdr delete replicateset

The cdr delete replicateset command deletes an exclusive or non-exclusive replicate set from the global catalog.

Syntax

```
cdr delete replicateset [ <Connect Option> (explicit id ) ] repl_set
```

Element	Description	Restrictions	Syntax
repl_set	Name of replicate set to delete. Can be the name of a derived replicate set.		Long Identifiers on page 256

Usage

The cdr delete replicateset command deletes the exclusive or non-exclusive replicate set *repl_set* from the global catalog.

The cdr delete replicateset command does not affect the replicates or associated data. When a replicate set is deleted, the individual replicates within the replicate set are unchanged.

Attention: Do not delete time-based exclusive replicate sets. Doing so might result in inconsistent data.

Attention: Avoid deleting a replicate set and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later.

Important: If you are creating a replicate set to replace the one you deleted, use the cdr check queue -qname=ctrlq command to make sure that the delete operation propagated to all the servers.

When you run the cdr delete replicateset command, an event alarm with a class ID of 69 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Example 1: Deleting a single replicate set

The following example connects to the database server specified by the **INFORMIXSERVER** environment variable and deletes the replicate set **accounts_set**:

```
cdr delete replset accounts_set
```

Example

Example 2: Deleting multiple replicate sets

The following example connects to database server **hoek** and deletes the replicate sets **accounts1_set** and **accounts2_set**:

```
cdr del replset -c hoek accounts1_set accounts2_set
```

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr define replicate on page 342

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

cdr check queue on page 310

Enterprise Replication Event Alarms on page 221

Related information

Altering multiple tables in a replicate set on page 190

Deleting a Replicate Set on page 175

Enterprise Replication Server administrator on page 18

cdr delete server

The cdr delete server disables a database server from participating in Enterprise Replication.

Syntax

cdr delete server [<Connect Option> (explicit id)] [**--force**] *server_group*

Element	Purpose	Restrictions	Syntax
server_group	A database server's `group` entry in its `INFORMIXSQLHOSTS` file and the global catalog.		Long Identifiers on page 256

The following table describes the option to the cdr delete server command.

Long Form	Short Form	Purpose
--force	-f	Remove an inactive Enterprise Replication server from the global catalog. You must use this option for standard servers that were converted from high-availability cluster servers.

Usage

The cdr delete server command disables a database server from to participating in Enterprise Replication. Use the --force option to disable an inactive replication server or to remove Enterprise Replication from a standard server that has been converted from a high-availability cluster server by the oninit -d standard command. You cannot delete a server that has non-root or leaf children under it. You must delete the children of a server before deleting the parent server.

The cdr delete server command generates event alarms with class IDs of 67 and 71, if the alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

To remove a server from an Enterprise Replication domain, run the cdr delete server command two times:

1. Run the command on the database server being removed, to disable it. This command does not propagate to other database servers in the domain.
2. Run the same command on a different server within the Enterprise Replication domain. This removes the disabled database server from the domain.

To remove an entire Enterprise Replication domain, run the cdr delete server command once on each replication server. The cdr delete server command performs the following tasks on each server:

1. Drops the Enterprise Replication connection to other hosts in the domain. A 25582 error and an operating system error might be printed to the online log.
2. Removes Enterprise Replication information, including delete tables and shadow columns.
3. Shuts down Enterprise Replication, if it is running.
4. Drops the local copy of the global catalog.

When you run the cdr delete server command on a different root server within an Enterprise Replication domain, the command performs the following tasks:

1. Deletes the command-specified database server from the global catalogs of all other servers in the domain.
2. Removes the command-specified database server from all participating replicates.
3. Purges all replication data destined for the command-specified database server from the send queues of all other servers in the domain.

Attention: Do not delete a replication server and immediately re-create it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the domain), failures might occur in the Enterprise Replication system at the time of the operation or later.

Important: If you are creating a replication server to replace the one you deleted, use the cdr check queue --qname=ctrlq command to make sure that the delete operation propagated to all the servers.

Example

Example 1: Removing a single database server from the domain

This example removes the database server **g_italy** from the Enterprise Replication environment. The commands are issued from **g_usa**:

```
cdr delete server -c italy g_italy
cdr delete server -c usa g_italy
```

The first command connects to database server **g_italy** and disables Enterprise Replication by removing the **syscdr** database and removing or stopping other Enterprise Replication components.

The second command performs the following actions:

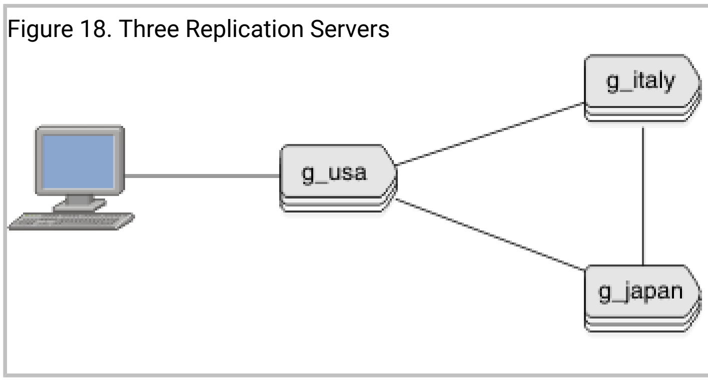
- Removes **g_italy** from the **g_usa** global catalog
- Drops the connection between **g_usa** and**g_italy**
- Removes **g_italy** from all participating replicates
- Purges the replication data destined for **g_italy** from send queues
- Broadcasts the delete command to all the other database servers in the Enterprise Replication domain so that the other servers can perform the same actions

Example

Example 2: Removing the whole domain

The following illustration shows a replication environment with three replication servers, **g_usa**, **g_italy**, and **g_japan**.

Figure 18. Three Replication Servers



To remove Enterprise Replication from every server in the domain, issue the cdr delete server command while connecting to each server. For example, from the computer containing the **g_usa** replication server, run these commands to remove Enterprise Replication and eliminate the domain:

```
cdr delete server -c italy g_italy
cdr delete server -c japan g_japan
cdr delete server g_usa
```

Example

Example 3: Removing Enterprise Replication from a high-availability server

In this example, the replication server group **g_usa** contains two servers that participate in a high-availability cluster: a primary (**usa_p**) and a secondary (**usa_s**). After **usa_s** is converted to a stand-alone server, the following command removes Enterprise Replication from it:

```
cdr delete server -c usa_s -f g_usa
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr disconnect server on page 385

cdr list server on page 402

cdr modify server on page 421

cdr resume server on page 442

cdr suspend server on page 473

cdr check queue on page 310

Enterprise Replication Event Alarms on page 221

Related information

Failover for High-availability clusters in an Enterprise Replication environment on page 90

Deleting a Replication Server on page 166

Enterprise Replication Server administrator on page 18

cdr delete shardCollection

The cdr delete shardCollection command deletes a sharding definition, and then stops data sharding.

Syntax

cdr delete shardCollection*definition_name* [<Connect Option> (explicit id)]

Element	Description	Restrictions
definition_name	The name of the sharding definition that is used for distributing data across multiple database servers.	Must be the name of an existing definition.

Usage

Use the cdr delete shardCollection command to delete a sharding definition, and then stop data sharding.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 99, 196, 229.

For information about these error codes, see Return Codes for the cdr Utility on page 262.

Example

Example

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers_1
   --type=delete --key=col2 --strategy=hash --versionCol=version
   shard_server_A
   shard_server_B
   shard_server_C
```

The following example deletes **collection_1**, and stops the sharding of table **customers_1**:

```
cdr delete shardCollection collection_1
```

Related reference

cdr define shardCollection on page 361

cdr change shardCollection on page 304

cdr list shardCollection on page 405

Related information

onstat -g shard command: Print information about the shard cache on page

Enabling sharding for JSON or relational data on page

Shard cluster management and monitoring on page 158

cdr delete template

The **cdr delete template** command deletes a template from the replication domain. It also deletes any underlying replicate sets associated with the template (these will exist if the template has been realized). No replicates are deleted.

Syntax

```
cdr delete template [ <Connect Option > (explicit id ) ] template
```

Element	Purpose	Restrictions	Syntax
template	Name of the template to delete.	The template must exist.	Long Identifiers on page 256

Usage

Use the **cdr delete template** command to delete the template definition and the replicate set realized from the template. Any replicates created by realizing the template to a database server are unaffected by this command.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following command deletes the template and replicate set **tem1**:

```
cdr delete template tem1
```

Related reference

cdr define template on page 367

cdr realize template on page 423

Enterprise Replication Event Alarms on page 221

Related information

Enterprise Replication Server administrator on page 18

cdr disable grid

The cdr disable grid command removes the authorization to run grid routines from users or servers.

Syntax

```
cdr disable grid [ <Connect Option > (explicit id ) ] --grid=grid_name [ --user=user_name ] [ --node=server_group ]
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
user_name	Name of the user.	Must be a user with authorization to run grid routines.	Long Identifiers on page 256

The following table describes the cdr disable grid options.

Long Form	Short Form	Meaning
--grid=	-g	Specifies the grid for which to revoke privileges.
--node=	-n	Specifies the servers on which to revoke privileges.
--user=	-u	Specifies the users to revoke privileges.

Usage

Use the cdr disable grid command to revoke the permission to run routines on the specified grid from the specified user or server that were granted by the cdr enable grid command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example revokes privileges from user **bill** on the **grid1** grid:

```
cdr disable grid --grid=grid1 --user=bill
```

The following example shows how to change the authorized server on the **grid1** grid from **gserv1** to **gserv2**:

```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2
```

Related information

Grid maintenance on page 130

Enterprise Replication Server administrator on page 18

cdr disable server

The cdr disable server command disables replication on a server.

Syntax

```
cdr disable server [ <Connect Option > (explicit id ) ] [ --local ] server_group
```

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server on which to disable replication.	Must be the name of an existing database server group in `sqlhosts`.	Long Identifiers on page 256

The following table describes the cdr disable server option.

Long Form	Short Form	Meaning
--local	-l	Disables the specified replication server. Must be run on both the replication server to disable and another replication server in the domain. Use this option if the connection is down between the replication server to disable and other replication servers.

Usage

Use the cdr disable server command when you need to temporarily stop replication and your replicates use the time stamp or delete wins conflict resolution rule.

When you run the cdr disable server command, the replication server is disabled and the rest of the replication domain is notified that the server is disabled.

If the replication server that you want to disable is not connected to the replication domain, you must run the cdr disable server command with the --local option on both the replication server to disable and another replication server in the domain. If the server on which you need to disable replication is currently offline, then run the cdr disable server command with the --local option on it after you restart it.

Disabling replication has the following effects:

- There is no connection between the disabled replication server and active replication servers.
- Transactions on the disabled replication server are not queued for replication
- Transactions on active replication servers are not queued for the disabled replication server.
- Control messages on active replication server are queued for the disabled replication server.
- Information about deleted rows on the disabled replication server is saved in delete tables.
- You can run only the following Enterprise Replication commands on the disabled replication server:
    - cdr enable server
    - cdr stop server
    - cdr delete server
    - cdr check replicateset with the --repair and the --enable options

You must synchronize the server after you enable replication on it. Shutting down and restarting the disabled replication server does not enable replication. You can both enable and synchronize a disabled replication server by running the cdr check replicateset command with the --repair and the --enable options. Alternatively, you can run the cdr enable server command and then synchronize the server.

Example

Example 1: Stopping replication on a connected server

The following command disables the server, **g_cdr1**, which is connected to the replication domain:

```
cdr disable server -c g_cdr1 g_cdr1
```

Example

Example 2: Stopping replication on a disconnected server

The following commands disable the replication server, **g_cdr1**, which is not connected to the replication domain:

```
cdr disable server -c g_cdr1 --local g_cdr1
cdr disable server -c g_cdr2 --local g_cdr1
```

The first command runs on the server **g_cdr1** and disables replication on it. The second command runs on the server **g_cdr2** and stops the other servers in the replication domain from queuing transactions for the server **g_cdr1**.

Related reference

cdr enable server on page 388

cdr check replicateset on page 325

Related information

Temporarily stopping replication on a server on page 164

Time stamp conflict resolution rule on page 42

Delete wins conflict resolution rule on page 48

Enterprise Replication Server administrator on page 18

cdr disconnect server

The **cdr disconnect server** command stops a server connection.

Syntax

cdr disconnect server [<Connect Option > (explicit id)] *server_group*

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server group to disconnect.	The database server group must be currently active in Enterprise Replication.	Long Identifiers on page 256

Usage

The **cdr disconnect server** command drops the connection (for example, for a dialup line) between *server_group* and the server specified in the **--connect** option. If the **--connect** option is omitted, the command drops the connection between *server_group* and the default database server (the one specified by the **INFORMIXSERVER** environment variable).

When you run the **cdr disconnect server** command, event alarms with class IDs of 54 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example drops the connection between the default database server (the one specified by the **INFORMIXSERVER** environment variable) and the server group **g_store1**:

```
cdr disconnect server g_store1
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr delete server on page 377

cdr list server on page 402

cdr modify server on page 421

cdr resume server on page 442

cdr suspend server on page 473

Related information

Enterprise Replication Server administrator on page 18

cdr enable grid

The cdr enable grid command authorizes users to run commands on the grid and designates servers from which grid commands can be run.

Syntax

cdr enable grid [<Connect Option > (explicit id)] **--grid**=*grid_name* [**--user**=*user_name*] [**--node**=*server_group*]

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
user_name	Name of the user.	Must have Connect privilege for databases on all the replication servers in the grid.	Long Identifiers on page 256

The following table describes the cdr enable grid options.

Long Form	Short Form	Meaning
--grid=	-g	Specifies the grid for which to provide privileges.
--node=	-n	Specifies the servers on which to provide privileges.
--user=	-u	Specifies the users to provide privileges.

Usage

Use the cdr enable grid command to control who can perform grid operations from which server in the grid. All the authorized users can run grid commands on all the authorized servers. The users must have Connect privilege for all databases on which they run grid routines on all the servers in the grid. You must authorize at least one user and one server to be able to run commands from the grid. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Authorizing more than one server from which to run grid commands can lead to conflicts between grid commands.

After you initially enable a grid, you can add authorized users and servers by running the cdr enable grid command with the appropriate options.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The following example authorizes the users **bill** and **tom** and the server **gserv1** to run grid routines on the grid named **grid1**:

```
cdr enable grid --grid=grid1 --user=bill --user=tom --node=gserv1
```

The following example adds the user **srini** to the list of authorized users for the **grid1** grid:

```
cdr enable grid --grid=grid1 --user=srini
```

The following example adds the server **gserv2** to the list of authorized servers for the **grid1** grid:

```
cdr enable grid --grid=grid1 --node=gserv2
```

Related reference

ifx_grid_connect() procedure on page 527

Related information

Creating a grid on page 129

Grid maintenance on page 130

Example of setting up a replication system with a grid on page 125

Enterprise Replication Server administrator on page 18

cdr enable server

The cdr enable server command enables replication on a replication server that was disabled by the cdr disable server command.

Syntax

```
cdr enable server [ <Connect Option > (explicit id ) ] [ --hub ] server_group
```

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server to enable.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

The following table describes the **cdr enable server** option.

Long Form	Short Form	Meaning
--hub	**-h**	Specifies that when replication on a hub server is enabled, replication on all its child servers is also enabled.

Usage

Use the cdr enable server command when you are ready to restart replication on a disabled replication server. After you enable replication, you must synchronize the server with the rest of the replication domain. Before synchronization is complete, the replicates on the newly enabled replication server have the Pending Sync attribute. For replicates with the Pending Sync attribute, ATS and RIS files are not created if transactions are aborted on this server. You can see the Pending Sync attribute of a replicate in the OPTIONS field of the output of the cdr list replicate command.

Example

Examples

The following command enables the disabled replication server, **g_cdr1**:

```
cdr enable server -c g_cdr1 g_cdr1
```

The following command enables the disabled replication server, **g_cdr1**, and its child servers:

```
cdr enable server -c g_cdr1 --hub g_cdr1
```

Related reference

cdr disable server on page 383

Related information

Restarting Replication on a Server on page 165

Enterprise Replication Server administrator on page 18

cdr error

The cdr error command manages the **syscdrerror** table and provides convenient displays of errors.

Syntax

`cdr error` [`<Connect Option >` (explicit id)] [{ `--seq=`*err_server*:*seqno* | `--prune"` [*first*,] *last*" | `--zap` | { | { `--follow | --all | --nomark` } } }]

Table 26. Elements for the cdr error command

Element	Purpose	Restrictions	Syntax
err_server	Name of database server group that holds the error table.	The server must be registered for Enterprise Replication.	Long Identifiers on page 256
first	Start date for a range.	You must provide a valid date and time.	Frequency Options on page 287
last	Ending date for range.	You must provide a later date and time than *first*.	Frequency Options on page 287
seqno	Sequence number of a specific error.	You must provide the number of an error in the error table.	Integer

Table 27. Options for the cdr error command

Long Form	Short Form	Meaning
(no options specified)		Print the current list of errors and then mark them as reviewed. Enterprise Replication does not display errors marked as reviewed.
--all	-a	Print all errors, including those already reviewed.
--follow	-f	Continuously monitor the error table.
--nomark	-n	Do not mark errors as reviewed.

Table 27. Options for the cdr error command (continued)

Long Form	Short Form	Meaning
--prune	-p	Prune the error table to those times in the range from *first* to *last*. If *first* is omitted, then all errors earlier than *last* are removed.
--seq	-s	Remove the (single) error specified by *server:seqno* from the error table.
--zap	-z	Remove all errors from the error table.

Usage

Run the cdr error command to examine replication errors. Sometimes a command succeeds on the server on which it is run but fails on one of the remote servers. For example, if you run the cdr define replicate command on server1, but the table name is misspelled on server2, the command succeeds on server1 and seems to complete successfully. You can use cdr error -c server2 to see why replication is failing.

The cdr error command also allows you to administer the **syscdrerror** table remotely. The **syscdrerror** table on each replication server contains errors for all replication servers, unless the replication server is a leaf node. The **syscdrerror** tables on leaf nodes do not contain errors for other replication servers. The reviewed flag indicates which errors are new errors while keeping the old errors in the table. For example, you can run cdr error periodically and append the output to a file.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following command shows the current list of errors on database server **hill**:

```
cdr error --connect=hill
```

After the errors are shown, Enterprise Replication marks the errors as reviewed.

The following command connects to the database server **lake** and removes from the error table all errors that occurred before the time when the command was issued:

```
cdr error -c lake --zap
```

The following command deletes all errors from the error table that occurred at or before 2:56 in the afternoon on May 1, 2008:

```
cdr error -p “2008-05-01 14:56:00?
```

The following command deletes all errors from the error table that occurred at or after noon on May 1, 2008 and before or at 2:56 in the afternoon on May 1, 2008:

```
cdr error -p “2008-05-01 14:56:00,2008-05-01 12:00:00?
```

Related information

Enterprise Replication Server administrator on page 18

cdr finderr

The **cdr finderr** command looks up a specific Enterprise Replication return code and displays the corresponding error text.

Syntax

```
cdr finderrER_return_code
```

Element	Purpose	Restrictions
ER_return_code	Enterprise Replication return code to look up.	Must be a positive integer.

You can also view the Enterprise Replication return codes in the file **$INFORMIXDIR/incl/esql/cdrerr.h**.

You can run this command from within an SQL statement by using the SQL administration API.

Related information

Return Codes for the cdr Utility on page 262

Enterprise Replication Server administrator on page 18

cdr list grid

The cdr list grid command shows information about a grid.

Syntax

```
cdr list grid [ <Connect Option > (explicit id ) ] [ [ { | --command=command_ID --verbose | --source=server_group --verbose
| --summary | --verbose | --nacks | --acks | --pending } ] grid_name ]
```

Element	Purpose	Restrictions	Syntax
command_ID	The ID of a specific command that was run from the grid.		An integer.
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers on page 256
server_group	Name of a database server group from which the command was run.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

The following table describes the cdr list grid options.

Long Form	Short Form	Meaning
--acks	-a	Displays the servers in the grid and the commands that succeeded on one or more servers.
--command=	-C	Displays the servers in the grid and the specified command.
--nacks	-n	Displays the servers in the grid and the commands that failed on one or more servers.
--pending	-p	Displays the servers in the grid and the commands that are in progress. A command can be pending because the transaction has not completed processing on the target server, the target server is down, or the target server was added to the grid after the command was run.
--source=	-S	Displays the servers in the grid and the commands that were run from the specified server.
--summary	-s	Displays the servers in the grid and the commands that were run on the grid.
--verbose	-v	Displays the servers in the grid, the commands that were run on the grid, and the results of the commands on each server in the grid.

Usage

Use the cdr list grid command to view information about servers in the grid, and about the commands that were run on servers in the grid.

If you run the cdr list grid command without any options or without a grid name, the output shows the list of grids.

Servers in the grid on which users are authorized to run grid commands are marked with an asterisk (*).

When you add a server to the grid, any commands that were previously run through the grid have a status of PENDING for that server. If you want to run previous grid commands on a new grid server, use the ifx_grid_redo() procedure. If you do not want to run previous grid commands on a new server, you can purge the commands by running the ifx_grid_purge() procedure.

When you run an SQL administration API command, the status of the grid command does not necessarily reflect whether the SQL administration API command succeeded. The grid command can have a status of ACK even if the SQL administration API command failed. The cdr list grid command shows the return codes of the SQL administration API commands. The task() function returns a message indicating whether the command succeeded. The admin() function returns an integer which if it is a positive number indicates that the command succeeded.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Examples

The examples in this section show the output of the cdr list grid command on a grid **grid1** that contains three servers: **cdr1**, **cdr2**, and **cdr3**.

Example

Example 1: Display grid members

The following command displays the members of the **grid1** grid:

```
cdr list grid grid1
```

The output of the previous command is:

```
Grid                Node                User
------------------ ------------------ ----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
```

This output shows that the grid contains three member servers and that the authorized user **bill** can run grid routines from the server **cdr1**.

Example

Example 2: Display verbose information about commands

The following command displays verbose information about a series of commands and their results on each server in the grid:

```
cdr list grid --verbose grid1
```

The output of the previous command is:

```
Grid                Node                User
------------------ ------------------ ----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3

Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
```

```
create table tab1 (col1 int, col2 int)
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3

Node:cdr1 Stmtid:3 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create procedure load(maxnum int)
define tnum int;
for tnum = 1 to maxnum
    insert into tab1 values (tnum, 1);
end for;
end procedure;
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3
```

This output shows each command and that all commands succeeded on servers **cdr1** and **cdr2** but are pending on the **cdr3** server because it is offline.

Example

Example 3: Display errors

In this example, the **cdr3** server already has a database with the same name as the database in the CREATE DATABASE statement: therefore, the CREATE DATABASE and CREATE TABLE statements fail. The following command displays information about commands run within the grid that resulted in an error:

```
cdr list grid --nacks grid1
```

The output of the previous command is:

```
Grid               Node               User
------------------ ------------------ ----------------
grid1              cdr1*              bill
                   cdr2
                   cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
NACK cdr3 2010-05-27 15:39:21 SQLERR:-330 ISAMERR:-100

Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create table tab1 (col1 int, col2 int)
NACK cdr3 2010-05-27 15:39:21 SQLERR:-310 ISAMERR:0
     Grid Apply Transaction Failure
```

This output shows the SQL and ISAM error codes associated with the failed statements.

Related reference

cdr define grid on page 338

cdr change grid on page 296

Related information

Grid maintenance on page 130

Rerunning failed grid routines on page 143

Example of setting up a replication system with a grid on page 125

Enterprise Replication Server administrator on page 18

cdr list replicate

The cdr list replicate command displays information about the replicates on the current server.

Syntax

```
cdr list replicate [ <Connect Option > (explicit id) ] [ { brief | full } ] [ replicate ]
```

Element	Purpose	Restrictions	Syntax
replicate	Name or ID of the replicates.	The replicates must exist.	Long Identifiers on page 256

Usage

The cdr list replicate command displays information about replicates (the full option). If no replicates are named, the command lists all replicates on the current server. If one or more replicates are named, the command displays detailed information about those replicates.

To display only replicate names and participant information, use the brief option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers obtain limited information about other database servers in the Enterprise Replication domain. Therefore, when cdr list replicate is run on a leaf server, it displays incomplete information about the other database servers.

The cdr list replicate command can be used while the replication server is in DDRBLOCK mode. Before you use the cdr list replicate command, you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the onspaces utility.

Output Description

The STATE field can include the following values.

Table 28. Values of the STATE field

Value	Description
Active	Specifies that Enterprise Replication captures data from the logical log and transmits it to participants
Definition Failed	Indicates that the replication definition failed on a peer server
Inactive	Specifies that no database changes are captured, transmitted, or processed
Pending	Indicates that a cdr delete replicate command ran and the replicate is waiting for acknowledgment from the participants
Quiescent	Specifies that no database changes are captured for the replicate or participant
Suspended	Specifies that the replicate captures and accumulates database changes but does not transmit any of the captured data

The `CONFLICT` field can include the following values.

Table 29. Values of the CONFLICT field

Value	Description
Deletewins	Specifies that the replicate uses the delete wins conflict-resolution rule
Ignore	Specifies that the replicate uses the ignore conflict-resolution rule
Timestamp	Specifies that the replicate uses the time stamp conflict-resolution rule
Procedure	Specifies that the replicate uses an SPL routine as the conflict-resolution rule

The `FREQUENCY` field can include the following values.

Table 30. Values of the FREQUENCY field

Value	Description
immediate	Specifies that replication occurs immediately
every *hh*:*mm*	Specifies that replications occur at intervals (for example, 13:20 specifies every thirteen hours and 20 minutes)

Table 30. Values of the FREQUENCY field

(continued)

Value	Description
at *day.hh:mm*	Specifies that replications occur at a particular time on a particular day (for example, 15.18:30 specifies on the 15th day of the month at 6:30 P.M.)

The `OPTIONS` field can include the following values.

Table 31. Values of the OPTIONS field

Value	Description
ats	Indicates that ATS files are generated if transactions fail to be applied at the target server.
firetrigger	Indicates that the rows that this replicate inserts fire triggers at the destination.
floatcanon	Indicates that floating-point numbers are replicated in machine-independent decimal representation.
floatieee	Indicates that floating-point numbers are replicated in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format.
fullrow	Indicates to replicate only changed columns and disable upserts.
ignoredel	Indicates that rows are retained if they are deleted on other nodes in the domain.
pendingsync	Indicates that the replication server was enabled with the cdr enable server command but that the participant is not yet synchronized with the rest of the domain. ATS and RIS files for this participant are not created if transactions are aborted.
ris	Indicates that RIS files are generated if transactions fail to be applied at the target server.
row	Indicates that the replicate uses row scope.
transaction	Indicates that the replicate uses transaction scope.
UTF8	Indicates that code set conversion between replicates is enabled.
TimeSeries	Indicates that the replicate includes a **TimeSeries** column.
alwaysRepLOBs	Indicates that large object columns are always included in replicated rows regardless of whether the large objects changed.

The `REPLTYPE` field can include the following values. If the `REPLTYPE` field does not show, the replicate is a classic replicate.

Table 32. Values of the REPLTYPE field

Value	Description
Master	Indicates that the replicate is defined as a master replicate.
Shadow	Indicates that the replicate is a shadow replicate. A shadow replicate can also be a master replicate.
Grid	Indicates that the replicate belongs to a grid replicate set.
Sendonly	Indicates that the participant only sends data.

The `PARENT REPLICATE` field shows only for shadow replicates. It shows the name of the replicate on which the shadow replicate is based.

Example

Examples

The following example displays a list of the replicates on the current server with full details:

```
cdr list replicate
```

The output from the command shows two replicates:

```
CURRENTLY DEFINED REPLICATES
---------------------------------------------
REPLICATE:        Repl1
STATE:            Inactive
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      bank:joe.teller
OPTIONS:          row,ris,ats
REPLTYPE:         Master

REPLICATE:        Repl2
STATE:            Inactive
CONFLICT:         Deletewins
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      bank:joe.account
OPTIONS:          row,ris,ats
REPLTYPE:         Master,Shadow
PARENT REPLICATE: Repl1
```

If the replicate belongs to a grid replicate set, the `REPLTYPE` field includes the value `Grid`.

```
CURRENTLY DEFINED REPLICATES
---------------------------------------------
REPLICATE:        grid_6553604_100_3
```

```
STATE:            Active ON:g_delhi
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      tdb:nagaraju.t1
OPTIONS:          row,ris,fullrow
REPLID:           6553605 / 0x640005
REPLMODE:         PRIMARY  ON:g_delhi
APPLY-AS:         INFORMIX ON:g_delhi
REPLTYPE:         Master,Grid
```

The `PARENT REPLICATE` field only shows if the replicate is a shadow replicate.

The following example displays a list of the replicates on the current server with brief details:

```
cdr list replicate brief
```

The output from the command shows the replicates:

```
REPLICATE   TABLE                                  SELECT
----------------------------------------------------------------
Repl1       bank@g_newyork:joe.teller              select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller         select * from joe.teller
Repl2       bank@g_portland:joe.teller             select * from joe.teller
Repl2       bank@g_atlanta:joe.teller              select * from joe.teller
```

The following example specifies the names of replicate:

```
cdr list repl brief Repl1
```

The output from the command shows information for the replicate:

```
REPLICATE   TABLE                                  SELECT
----------------------------------------------------------------
Repl1       bank@g_newyork:joe.teller              select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller         select * from joe.teller
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

cdr swap shadow on page 475

cdr list replicateset on page 400

Related information

Adding an existing replicate to a grid replicate set by using cdr change replicateset on page 134

Viewing grid information on page 131

Preventing Memory Queues from Overflowing on page 212

cdr list replicateset

The **cdr list replicateset** command displays information about the replication sets defined on the current server.

Syntax

```
cdr list replicateset [ <Connect Option > (explicit id ) ] [ repl_set ]
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set.	The replicate set must exist.	Long Identifiers on page 256

Usage

The **cdr list replicateset** command displays a list of the replicate sets that are currently defined. To list the information about each of the replicates within the replicate set, use **cdr list replicateset** *repl_set*.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list replicateset** is executed against a leaf server, it displays incomplete information about the other database servers.

If you specify the name of a grid replicate set, the command displays the names of the replicates that were automatically created through the grid and any replicates manually added to the grid replicate set. The name of the grid replicate set is the same as the name of the grid.

The **cdr list replicateset** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list replicateset** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

Example

Examples

The following example displays a list of the replicate sets on the current server:

```
cdr list replicateset
```

The following output might result from the previous command:

```
Ex T REPLSET                  PARTICIPANTS
-----------------------------------------------
N  Y  g1                         Repl1, Repl4
N  Y  g2                         Repl2, Repl3, Repl5
```

The `Ex` field shows whether the replicate set is exclusive. The `T` field shows whether the replicate set was created from a template.

This example displays information for all the replicates in the replicate set **g1**:

```
cdr list replset g1
```

The following output might result from the previous command:

```
REPLICATE SET:g1 [Exclusive]
CURRENTLY DEFINED REPLICATES
----------------------------------------------------------------------
REPLICATE:     Repl1
STATE:         Inactive
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   bank:arthur.account
OPTIONS:       row,ris,ats
REPLTYPE:      Master

REPLICATE:     Repl4
STATE:         Inactive
CONFLICT:      Deletewins
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   bank:arthur.teller
OPTIONS:       row,ris,ats
REPLTYPE:      Master
```

The information supplied for each replicate is the same as the information provided by the **cdr list replicate** command.

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr define replicate on page 342

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

cdr list replicate on page 395

Related information

Example of setting up a replication system with a grid on page 125

Viewing grid information on page 131

cdr list server

The **cdr list server** command displays a list of the Enterprise Replication servers that are visible to the server on which the command is run.

Syntax

```
cdr list server [ <Connect Option > (explicit id ) ] [ server_group ]
```

Element	Purpose	Restrictions	Syntax
server_group	Name of the server group.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr list server** command displays information about servers. You do not need to be user **informix** to use this command; any user can run it.

The **cdr list server** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list server** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

When no server-group name is given, the **cdr list server** command lists all database server groups that are visible to the current replication server.

In hierarchical topology, leaf servers only have information about their parent database servers in the Enterprise Replication domain. Therefore, when **cdr list server** is executed against a leaf server, it displays incomplete information about the other database servers.

Output Description

The SERVER and ID columns display the name and unique identifier of the Enterprise Replication server group.

The STATE column can have the following values.

Value	Description
Active	The server is active and replicating data.
Deleted	The server has been deleted; it is not capturing or delivering data and the queues are being drained.
Disabled	The server is disabled. It is not capturing or delivering data, but its delete tables are being maintained.

Value	Description
Quiescent	The server is in the process of being defined.
Suspended	Delivery of replication data to the server is suspended.

The STATUS column can have the following values.

Value	Description
Connected	The connection is active.
Connecting	The connection is being established.
Disconnect	The connection was explicitly disconnected.
Disconnected will attempt reconnect	The connection was disconnected but is being reattempted.
Dropped	The connection was disconnected due to a network error because the server is unavailable.
Error	The connection was disconnected due to an error (check the log and contact customer support, if necessary).
Failed	The connection attempt failed.
Local	Identifies that this server is the local server as opposed to a remote server.
Timeout	The connection attempt has timed out, but will be reattempted.

The QUEUE column displays the size of the queue for the server group.

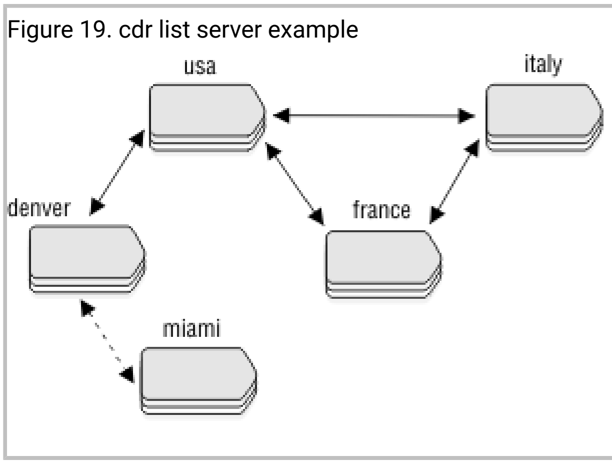
The CONNECTION CHANGED column displays the most recent time that the status of the server connection was changed.

Example

Examples

In the following examples, **usa**, **italy**, and **france** are root servers, **denver** is a nonroot server, and **miami** is a leaf server. The **usa** server is the parent of **denver**, and **denver** is the parent of **miami**.

Figure 19. cdr list server example



When the **cdr list server** command includes the name of a database server group, the output displays the attributes of that database server. The following commands and example output illustrate how the **cdr list server** command displays server information.

In this example, the server **g_usa** generates ATS and RIS files in XML format, has an idle time out of 15 seconds, and is a hub server.

```
cdr list server g_usa

NAME      ID     ATTRIBUTES
-----------------------------------------------------
g_usa     1      atsrisformat=xml timeout=15 hub
```

In this example, the **g_denver** server shows the **g_usa** server as its root server.

```
cdr list server -c denver g_denver
NAME      ID     ATTRIBUTES
-----------------------------------------------------
g_denver  27     root=g_usa
```

In this example, the attributes of the **g_denver** server are shown from the perspective of the **italy** server. The **g_denver** server has the **g_usa** server as its root server and uses the **g_usa** server to forward replicated transactions between it and the **italy** server.

```
cdr list server -c italy g_denver

NAME      ID     ATTRIBUTES
-----------------------------------------------------
g_denver  27     root=g_usa forward=g_usa
```

In this example, the **g_miami** server shows the **g_denver** server as its root server and that it is a leaf server.

```
cdr list server g_miami

NAME      ID     ATTRIBUTES
```

```
-----------------------------------------------------
g_miami   4      root=g_denver leaf
```

The following example shows possible output for the **cdr list server** command if no server groups are specified:

```
cdr list server
SERVER         ID STATE     STATUS     QUEUE CONNECTION CHANGED
--------------------------------------------------------------
g_denver       1 Active   Local      0
g_miami        2 Active   Connected  0     Mar 19 13:48:44
g_usa          3 Active   Connected  0     Mar 19 13:48:40
g_france       4 Active   Connected  0     Mar 19 13:48:41
g_italy        5 Active   Connected  0     Mar 19 13:48:45
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr delete server on page 377

cdr disconnect server on page 385

cdr modify server on page 421

cdr resume server on page 442

cdr start on page 443

cdr suspend server on page 473

cdr view on page 486

cdr resume replicate on page 440

cdr resume replicateset on page 441

Related information

Preventing Memory Queues from Overflowing on page 212

cdr list shardCollection

The cdr list shardCollection command displays the sharding definition for all database servers in a shard cluster.

Syntax

cdr list shardCollection*definition_name* [<Connect Option> (explicit id)]

Element	Description	Restrictions
definition_name	The name of the sharding definition that is used for distributing data across multiple database servers.	Must be the name of an existing definition.

Usage

The cdr list shardCollection command displays the sharding definition for database servers in a shard cluster.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 99, 196, 229.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Example: Output for a sharding definition that uses consistent hash-based sharding

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

The following example shows output when the cdr list shardCollection command is run on a database server in the shard cluster. Each shard server has three hashing partitions.

Figure 20. Output when the cdr list shardCollection is run on a shard server that uses consistent hash-based sharding.

```
Shard Collection:shrdb Version:0 type:consistent hash key:b
Version Column:column_3
Table:db_1:john.customers
g_shard_server_1     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
g_shard_server_2     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
g_shard_server_3     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950)
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 40
```

Example

Example: Output for a sharding definition that uses hash-based sharding

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 database_1:john.customers_1
   --type=delete --key=col2 --strategy=hash --versionCol=column_3
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following example shows output when the cdr list shardCollection command is run on a database server in the shard cluster.

Figure 21. Output when the cdr list shardCollection is run on a shard server that uses hash-based sharding.

```
Shard Collection:collection_1 Version:0 type:hash key:col2
Version Column:column_3
Table:database_1:john.customers_1
g_shard_server_A      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  = 0
g_shard_server_B      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (1, -1)
g_shard_server_C      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (2, -2)
g_shard_server_D      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (3, -3)
```

Example

Example: Output for a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_2 database_2:joe.customers_2
   -t delete -k state -s expression -v column_3
   g_shard_server_F "IN ('AL','MS','GA')"
   g_shard_server_G "IN ('TX','OK','NM')"
   g_shard_server_H "IN ('NY','NJ')"
   g_shard_server_I  remainder
```

The following example shows output when the cdr list shardCollection command is run on a database server in the shard cluster.

Figure 22. Output when the cdr list shardCollection is run on a shard server that uses expression-based sharding.

```
Shard Collection:collection_2 Version:0 type:expression key:state
Version Column:column_3
Table:database_2:joe.customers_2
g_shard_server_F      state IN ('AL','MS','GA')
g_shard_server_G      state IN ('TX','OK','NM')
g_shard_server_H      state IN ('NY','NJ')
g_shard_server_I      not ( (state IN ('AL','MS','GA') ) or  (state
IN ('TX','OK','NM') ) or  (state IN ('NY','NJ') ))
```

Example

Example: Output for a sharding definition that was modified

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_3 database_3:tony.customers_3
   -t keep -k bson_value_lvarchar(data,'year') -s expression -v column_3
   g_shard_server_J "BETWEEN 1970 and 1979"
   g_shard_server_K "BETWEEN 1980 and 1989"
   g_shard_server_L "BETWEEN 1990 and 1999"
   g_shard_server_M  remainder
```

The sharding definition is then modified by the following command:

```
cdr change shardCollection collection_3 -a
   g_shard_server_N "BETWEEN 2000 and 2009"
```

The sharding definition is then modified a second time:

```
cdr change shardCollection collection_3 -d g_shard_server_J
```

The following example shows output when the cdr list shardCollection command is run on a database server in the shard cluster. The `Version` value increments with each cdr change shardCollection command that successfully runs on **collection_3**.

Figure 23. Output when the cdr list shardCollection is run on a shard server that has a modified sharding definition.

```
Shard Collection:collection_3 Version:2 type:expression
key:bson_value_lvarchar(data,'year') Version Column:column_3
Table:database_3:tony.customers_3
g_shard_server_K      bson_value_lvarchar(data,'year') BETWEEN 1980 and 1989
g_shard_server_L      bson_value_lvarchar(data,'year') BETWEEN 1990 and 1999
g_shard_server_N      bson_value_lvarchar(data,'year') BETWEEN 2000 and 2009
g_shard_server_M      not((bson_value_lvarchar(data,'year') BETWEEN 1980 and 1989)
or (bson_value_lvarchar(data,'year') BETWEEN 1990 and 1999) or (bson_value_lvarchar
(data,'year') BETWEEN 2000 and 2009))
```

Related reference

cdr define shardCollection on page 361

cdr change shardCollection on page 304

cdr delete shardCollection on page 380

Related information

onstat -g shard command: Print information about the shard cache on page

Enabling sharding for JSON or relational data on page

Viewing shard-cluster participants on page

Shard cluster management and monitoring on page 158

cdr list catalog

The cdr list catalog command lists the commands that created the specified replication objects.

Syntax

cdr list catalog [<Connect Option > (explicit id)] [**--quiet**] [{ { | **--servers** | **--replicates** | **--replicatesets** | **--templates** | **--realizetemplates** | **--grids** } | **--all** }]

The following table describes the options to the cdr list catalog command.

Long Form	Short Form	Meaning
--all	-a	Lists all definition commands. Default.

Long Form	Short Form	Meaning
--grids	-g	Lists cdr create grid commands.
--quiet	-q	Lists the commands without headings.
--realizetemplates	-z	Lists cdr realize template commands.
--replicates	-r	Lists cdr define replicate commands.
--replicatesets	-e	Lists cdr define replicateset commands.
--servers	-s	Lists cdr define server commands.
--templates	-t	Lists cdr define template commands.

Usage

Run the cdr list catalog command to show replication definition commands. You can use the list of commands to easily duplicate a system for troubleshooting or moving a test system into production.

Example: List template commands

The following command lists the cdr define template and the cdr realize template commands:

```
$ cdr list cat -t

#
# cdr define template commands.
#

cdr define template temp1 --conflict=ignore  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp2 --conflict=always  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp3 --conflict=timestamp  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp4 --conflict=timestamp --floatieee --ats --ris
        --alwaysRepLOBs=y --UTF8=y --master=g_cat_cdr1 --database=catdb
        informix.tab1 informix.tab2 informix.tab4
```

Example: List server commands

The following command lists the cdr define server commands:

```
$ cdr list cat -s

#
# cdr define server commands.
#

cdr define server --init g_cat_cdr1
cdr define server --connect=g_cat_cdr2 --sync=g_cat_cdr1 --init g_cat_cdr2

cdr define server --connect=g_cat_cdr3 --ats=/usr/informix/ats
       --ris=/usr/informix/ris --sync=g_cat_cdr1 --init g_cat_cdr3

cdr define server --connect=g_cat_cdr4 --sync=g_cat_cdr1
       --init g_cat_cdr4
```

cdr list template

The **cdr list template** command displays information about the templates on the server on which the command is run.

Syntax

cdr list template [<Connect Option > (explicit id)] [*template*] [{ **BRIEF** | **FULL** }]

Element	Purpose	Restrictions	Syntax
template	Name of the template.	The template must exist.	Long Identifiers on page 256

Usage

The **cdr list template** command displays information about templates. If no templates are named, the command lists all templates in the Enterprise Replication domain. If one or more templates are named, the command displays the names, database names, and table names for those templates.

To display detailed information for your templates, use the **FULL** option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list template** is executed against a leaf server, it displays incomplete information about the other database servers.

The **cdr list template** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list template** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the onspaces utility.

Example

Examples

The following example displays detailed information about the templates on the current server:

```
cdr list template
```

The output from the previous command might be the following:

```
TEMPLATE            DATABASE            TABLES
==============================================
tem1                newcars             table1
```

```
                    newcars             table2
                    newcars             table3
tem2                carparts            table1
                    carparts            table3
```

The following example displays detailed information about the template **tem1**:

```
cdr list template tem1
```

The output from the previous command might be the following:

```
CURRENTLY DEFINED TEMPLATES
===========================
TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_1_table1
OWNER:        pravin
TABLE:        table1

TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_2_table2
OWNER:        pravin
TABLE:        table2

TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_3_table3
OWNER:        pravin
TABLE:        table3
```

Related reference

cdr define template on page 367

cdr realize template on page 423

cdr migrate server

The cdr migrate server command automates data migration task between two or more servers. This command also automates setting up of Enterprise Replication between two Informix server instances.

Data migration options

- Static mode-offline data migration
  - Create storage spaces using storage pool
  - Migrate schema (except referential integrity) and data in parallel
  - Build referential integrity

- Dynamic mode – online data migration
    - Create storage spaces using storage pool
    - Migrate schema (except referential integrity) and data in parallel
    - Resynchronize data using Enterprise Replication
    - Build referential integrity

The cdr migrate server command automates tasks such as:

- Define Enterprise Replication domain between the two servers.
- Add ERKEY columns for the tables that does not have primary key or unique index columns.
- Create required storage spaces for the databases
- Apply database schema from source server to target server
- Create replicate definitions
- Synchronize data between source and target server instances
- Build referential integrity

Supported phases

1. define_er: Define ER
2.
    a. add_erkey: Add ERKEY to source schema
    b. add_replcheck : Add REPLCHECK column and index to source schema
3. create_spaces: Create storage spaces using storagepool
4.
    a. create_schema_loaddata: Multi phase replay of schema and data
        - Phased migration of schema and data:
            - Phase 1:
                - Migrate database schema except indexes, primary key, unique and referential constraints
                - Create all tables as “RAW? tables
            - Phase 2:
                - Start multiple parallel jobs for data load and index builds
                - Each job includes: make sure table is a raw table, Load data using “insert into … select * from …? ISTAR query, Build indexes, primary key and unique key constraints
            - Phase 3:
                - Build referential constraints
    b. create_schema_nodata : Create database schema
    c. create_schema_loaddata_nori : Create database schema without referential integrity and load data
5. create_replicates: Create replicate, replicate set and grid definitions
6. sync_data: Synchronize data
7. add_ri: Add referential integrity
8. all: Execute all phases (with phase create_schema_loaddata_nori)
9. static: Execute create_spaces and create_schema_loaddata phases
10. dynamic: Execute all phases similar to ‘all’

If you decide to execute the above phases individually, then phase order need to be maintained.

Prerequisites to run cdr migrate server command

1. To auto create required storage spaces, storage pool is a requirement at source and target servers.
2. SQLHOSTS files at both source and target server must be already configured with ER group information.
3. Trusted host configuration must be already established between source and target servers.
4. Source server must be 11.70xC1 or higher version.

Restrictions

1. Cannot mix multiple database code-sets in same data migration command.
2. Parallel data load using ISTAR query do not support tables with smart large objects(BLOB and CLOB), user created UDTs and collection datatypes.
    - Requires customization to unload/load data using external tables or using any other supported unload/load commands.
3. Tenant databases and database sharding are not supported.
4. Cannot use this tool for database code-set migration

Command syntax

```
cdr migrate server -s source -t target -p phase [-d database] [--exec]
```

-s --source Source server name

-t --target Target server name

-p --phase migration phase

-e --exec Execute commands. Default: print only.

-d --database Database to replicate

-T --receiveonly Setup oneway replication from source to target

('create_replicates' phase)

-r --checkrepair Synchronize data using 'cdr check'

instead of 'cdr sync'('sync_data' phase)

-g --grid Y/N Enable/Disable grid. Default:Y('create_replicates' phase)

-A --atsdir ATS files directory path('define_er' phase)

-R --risdir ATS files directory path('define_er' phase)

-P --parallelsync Number of table sync jobs to run in parallel('sync_data' phase)

-f --outfile <file path> Output file for commands

-E --exclude db:owner.tab specify table to exclude

Example

Examples

To print commands to stdout:

```
cdr migrate server   -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase all
```

To execute commands:

```
cdr migrate server   -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase all  --exec
```

To run 'define_er' phase:

```
cdr migrate server  -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase define_er --atsdir=/work --risdir=/work
```

cdr modify grid

The **cdr modify grid** command modifies grid attributes.

Syntax

cdr modify grid [<Connect Option > (explicit id)] *grid_name* { **--enablegridcopy | --disablegridcopy** } [*Server*]

Element	Purpose	Restrictions	Syntax
grid_name	Name of a grid to modify		
Server	Name of a server to modify		

The following table describes the options to **cdr modify grid**.

Long Form	Short Form	Meaning
--enablegridcopy	**-E**	Enables the specified server to perform grid copy functions.
--disablegridcopy	**-D**	Disables the specified server from performing grid copy functions.

Usage

The **cdr modify grid** command modifies the attributes of one or more servers in a grid. If the command does not specify a server, the changes apply to all servers in the grid.

The --enablegridcopy option is used only if a grid was created using Informix® version 11.70 and then upgraded to Informix® version 12.10 or later.

Grids created using Informix® version 11.70 and earlier cannot copy external files to a grid. If you upgrade servers in a grid from 11.70 to 12.10, and you want to copy external files to servers in the grid, you must enable the ability to copy external files by running the cdr modify grid command with the --enablegridcopy option. Similarly, before reverting from Informix® version 12.10 to an earlier version of Informix®, you must disable the ability to copy external files by running the cdr modify grid command with the --disablegridcopy option.

It is not necessary to run the cdr modify grid command if your grid was created using Informix® version 12.10 or later.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example enables copying external files on all servers in the grid named **grid1**:

```
cdr modify grid  grid1 --enablegridcopy
```

The following example enables copying external files on the server **g_serv1** in the grid named **grid1**:

```
cdr modify grid  grid1 --enablegridcopy g_serv1
```

The following example disables copying external files on all servers in the grad named **grid1**:

```
cdr modify grid  grid1  --disablegridcopy
```

The following example disables copying external files for the server **g_serv1** in the grid named **grid1**:

```
cdr modify grid  grid1  --disablegridcopy g_serv1
```

Related information

Enterprise Replication Server administrator on page 18

cdr modify replicate

The cdr modify replicate command modifies replicate attributes.

Syntax

```
cdr modify replicate [ <Connect Option > (explicit id ) ] [ --name=n ] [ { | <Conflict Options > (explicit id ) | <Scope Options >
(explicit id ) | <Frequency Options > (explicit id ) | <Special Options > (explicit id ) } ] replicate [ participant ]
```

Element	Purpose	Restrictions	Syntax
participant	Name of a participant in the replication.	The participant must be a member of the replicate.	Participant and participant modifier on page 257
replicate	Name of the replicate to modify.	The replicate name must exist.	Long Identifiers on page 256

The following table describes the option to cdr modify replicate.

Long Form	Short Form	Meaning
--name=n	-n n	Removes the name verification attribute from a master replicate.

Special Options

Special Options

```
“[{| --ats{y | n}| --ris{y | n}| --firetrigger{y | n}| --fullrow{y | n}| --ignoredel={y |
n}| --serial| --UTF8={y | n}| --alwaysRepLOBs={y | n}}]”
```

Table 33. Special options for cdr modify replicate.

Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • --alwaysRepLOBS=n: Default. Columns that contain unchanged large objects are not replicated. • --alwaysRepLOBS=y: Columns that contain large objects are always included in replicated rows.
--ats y or --ats n	-A y or -A n	Activates (y) or deactivates (n) aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
--firetrigger y or	-T y or	Causes the rows that are inserted by the replicate to fire (y) or not fire (n) triggers at the destination.

Table 33. Special options for cdr modify replicate. (continued)

Long Form	Short Form	Meaning
--firetrigger n	-T n	
--fullrow y or --fullrow n	-f y or -f n	Specifies to (y) replicate the full row and enable upserts or (n) replicate only changed columns and disable upserts.
--ignoredel=	-D	Specifies whether to retain deleted rows on other nodes: • --ignoredel=y = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify deletewins as the conflict resolution rule. • --ignoredel=n = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris y or --ris n	-R y or -R n	Activates (y) or deactivates (n) row-information spooling for replicate row data that fails conflict resolution or encounters replication-order problems.
--serial	-s	Specifies that replicated transactions for the replicate are applied serially instead of in parallel.
--UTF8=		Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets. • --UTF8=y Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types, such as user-defined types or DataBlade® data. No attempt is made to convert character data that is contained within opaque data types. You cannot use --UTF8=y

Table 33. Special options for cdr modify replicate. (continued)

Long Form	Short Form	Meaning
		for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade® module data types. • -UTF8=n Indicates that code set conversion is ignored.

Usage

The cdr modify replicate command modifies the attributes of a replicate or of one or more participants in the replicate. You can also change the mode of a participant. If the command does not specify participants, the changes apply to all participants in the replicate.

To add or delete a participant, use the cdr change replicate command.

If you change the conflict resolution rule with cdr modify replicate, you must also specify the scope with the --scope option, even if you are not changing the scope.

The attributes for cdr modify replicate are the same as the attributes for cdr define replicate, with the following exceptions:

- You cannot change the machine-independent decimal representation (--floatcanon) or IEEE floating point (--floatieee) formats.
- You cannot change the conflict resolution from ignore to a non-ignore option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-ignore conflict resolution option to ignore.

  However, you can change from time stamp resolution to SPL routine resolution or from SPL routine resolution to time stamp.
- The --ats, --ris, --firetrigger, and --fullrow options require a yes (y) or no (n) argument.

When you run the cdr modify replicate command, an event alarm with a class ID of 63 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example modifies the frequency attributes of replicate **smile** to replicate every five hours:

```
cdr modify repl --every=300 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate daily at 1:00 A.M.:

```
cdr modify repl -a 01:00 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate on the last day of every month at 5:00 A.M., to generate ATS files, and not to fire triggers:

```
cdr modify repl -a L.5:00 -A y -T n smile
```

The following example changes the mode of the first participant that is listed to receive-only and the mode of the second to primary:

```
cdr mod repl smile "
R db1@server1:antonio.table1"
 \
                   "
P db2@server2:carlo.table2"
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

Enterprise Replication Event Alarms on page 221

Participant and participant modifier on page 257

Related information

Enabling ATS and RIS File Generation on page 200

Creating Strict Master Replicates on page 102

Frequency Options on page 287

Enterprise Replication Server administrator on page 18

Change replicate attributes on page 168

cdr modify replicateset

The **cdr modify replicateset** command modifies all the replicates in a replicate set.

Syntax

cdr modify replicateset [{| <Connect Option > (explicit id) | <Frequency Options > (explicit id) }] *repl_set*

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to modify.	The replicate set must exist.	Long Identifiers on page 256

Usage

The **cdr modify replicateset** command modifies the attributes of all the replicates in the replicate set *repl_set*. To add or delete replicates from a replicate set, use the **cdr change replicateset** command.

You cannot change whether a replicate set is exclusive or not.

When you run the **cdr modify replicateset** command, an event alarm with a class ID of 64 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default server (the server specified by the **INFORMIXSERVER** environment variable) and modifies the replicate set **sales_set** to process replication data every hour:

```
cdr mod replset --every=60 sales_set
```

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr define replicate on page 342

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

Enterprise Replication Event Alarms on page 221

Related information

Frequency Options on page 287

Enterprise Replication Server administrator on page 18

cdr modify server

The cdr modify server command modifies the Enterprise Replication attributes of a database server.

Syntax

```
cdr modify server [ <Connect Option > (explicit id ) ][ --idle =timeout ][ --mode { primary | readonly | sendonly }][ --
ats=ats_dir ][ --ris=ris_dir ][ --atsrisformat = { text | xml | both }] server_group
```

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to modify.	The database server group must be defined in Enterprise Replication.	
timeout	Idle timeout for this server.	Must be an integer number of minutes. 0 indicates no timeout. The maximum value is 32,767.	Integer.
ats_dir	Name of Aborted Transaction Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of `/dev/null` (UNIX™) or `NUL` (Windows™) prevents ATS file generation.	Follows naming conventions on your operating system.
ris_dir	Name of the Row Information Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of `/dev/null` (UNIX™) or `NUL` (Windows™) prevents RIS file generation.	Follows naming conventions on your operating system.

The following table describes the options to cdr modify server.

Long Form	Short Form	Meaning
--ats=	-A	Activates aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
-atsrisformat=	-X	Specifies the format of ATS and RIS files: • text: ATS and RIS files are generated in standard text format. • xml: ATS and RIS files are generated in XML format. • both: ATS and RIS files are generated in both standard text format and XML format.

Long Form	Short Form	Meaning
--idle=	-i	Causes an inactive connection to be terminated after *timeout* minutes. If time-out is 0, the connection does not time out. The default value is 0.
--mode	-m	Changes the mode of all replicates using this server: • primary: The participant both receives and sends replicated data. • readonly: The participant only receives replicated data and does not send replicated data. • sendonly: The participant only sends replicated data and does not receive replicated data. **Note:** The -m option only affects replicates whose conflict resolution is ignore.
--ris=	-R	Activates row-information spooling for replicate-row data that fails conflict resolution or encounters replication-order problems.

Usage

The cdr modify server command modifies the replication server *server_group*.

When you run the cdr modify server command, an event alarm with a class ID of 70 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the database server **paris** and modifies the idle time-out of server group **g_rome** to 10 minutes. ATS files are generated into the directory `/cdr/atsdir` in both text and XML format.

```
cdr modify server -c paris -i 10 -A /cdr/atsdir \
-X both g_rome
```

The following example connects to the default database server and sets the modes of all participants on **g_geometrix** to primary:

```
cdr mod ser -m p g_geometrix
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr delete server on page 377

cdr disconnect server on page 385

cdr list server on page 402

cdr resume server on page 442

cdr suspend server on page 473

Enterprise Replication Event Alarms on page 221

Related information

Enabling ATS and RIS File Generation on page 200

Disabling ATS and RIS File Generation on page 212

Monitor and troubleshooting Enterprise Replication on page 197

Customizing the Replication Server Definition on page 99

Enterprise Replication Server administrator on page 18

Data consolidation on page 36

Primary-Target Data Dissemination on page 35

Modify server attributes on page 161

cdr realize template

The cdr realize template command creates the replicates, replicate set, and participant tables as specified in a template, and then synchronizes data on all or a subset of the database servers within the replication domain.

Syntax

```
cdr realize template [ <Connect Option > (explicit id ) ] template [ --syncdatasource=data_server <Synchronization Options> ]
[ { --verify | [ --autocreate [ --dbspace= dbspace ] ] } ] [ --target ] [ --mode = { send_only | receive_only } ] [ database@ ]
server_group [ --applyasowner ]
```

Synchronization Options

```
“ [ --extratargetrows= { delete | keep | merge } ] ”

“ [ --foreground [ --memadjust= size { K | M } ] ] ”
```

Element	Purpose	Restrictions	Syntax
database	Name of the database that includes the table to be replicated.	The database server must be in an Enterprise Replication domain.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
dbspace	The name of the dbspace for tables.	The dbspace must exist on all the database servers listed. If you do not specify a dbspace name and new tables are created, they are created in the default dbspace.	
server_group	Name of the database server group that includes the server to connect to.	The database server group name must be the name of an existing Enterprise Replication server group in `sqlhosts`.	Long Identifiers on page 256
*size*K or *size* M	Size, in either kilobytes (K) or megabytes (M), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
template	The name of the template.	The template must exist. Use the cdr define template command to create the template.	Long Identifiers on page 256

The following table describes the special options to cdr realize template.

Long Form	Short Form	Meaning
--applyasowner	-o	Specifies that any tables created when you realize the template are owned by the owner of the source tables. By default, the tables are owned by the user **informix**.
--autocreate	-u	Specifies that if the tables in the template definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. **Note:** Tables that are created with autocreate do not automatically include non-replicate key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. You must manually create these objects.
--dbspace=	-D	Specifies the dbspace in which the automatically created objects are placed. If not specified, then the default dbspace is used.

Long Form	Short Form	Meaning
--extratargetrows=	-e	Specifies how to handle rows that are found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • delete: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • keep: retain rows on the target servers • merge: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	-F	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	-J	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--mode=	-m	Specifies whether the participant either only sends or only receives replicated data: • send_only (S): The participant only sends replicated data and does not receive replicated data. • receive_only (R): The participant only receives replicated data and does not send replicated data.
--syncdatasource=	-S	Specifies which server is the source of the data that is used to synchronize all the other servers that are listed in the cdr realize template command. The server that is listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.
--target	-t	Specifies that all of the servers that are listed in the command become receive-only servers, including the source server, unless the template is already realized on the source server. If you use this option, you must run the cdr realize template command twice: once to realize the template on the source server and other

Long Form	Short Form	Meaning
		primary servers, and again to realize the template on receive-only servers.
--verify	-v	Specifies that the cdr realize template command verifies that the database, tables, column data types are correct on all listed servers, but does not realize the template.

Usage

Before you can use the cdr realize template command, you must define Enterprise Replication servers by running the cdr define server command and define the template by running the cdr define template command. Create the database to be replicated on all database servers in the replication domain. However, only the database on the synchronization data source server must be populated with data.

All specified servers must be online and the cdr utility must be able to connect to each server.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The cdr realize template command performs the following tasks:

- If you specify the --autocreate option, creates database tables on the target servers.

  Recommendation: If you use --autocreate, specify a dbspace name. If you do not specify a dbspace name, tables are created in the root dbspace, which is not recommended.

- If you specify the --verify option, verifies the database, tables, column data types, and replication keys on all participating servers; however, the template is not realized.
- If you specify the --syncdatasource option, synchronizes the data from the source database with the databases specified by this command. If you specify the --foreground option, runs synchronization as a foreground process. If you specify the --memadjust option, increases the size of the send queue from the value of the CDR_QUEUEMEM configuration parameter.

  If you are running this command with the --syncdatasource option as a DBSA, you must have certain permissions granted to you on several system tables in the syscdr database. For more information, see Preparing for Role Separation (UNIX) on page 80.
- Verifies the database and table attributes to ensure that replication can be performed on each database.
- Creates replicates as master replicates on all servers.
- Creates a replicate set for the new replicates.
- Starts the replicates on all servers.

The replicates and replicate set created from a template have generated names. Use the cdr list template command to see the names of the replicates and replicate set associated with a particular template.

You can run this command from within an SQL statement by using the SQL administration API.

You can run the cdr check queue --qname=cntrlq command to wait for the cdr realize template command to be applied at all Enterprise Replication servers before you run the data synchronization task.

Example

Examples

The following example illustrates the cdr realize template command:

```
cdr realize template tem1 -c detroit\
new_cars@detroit new_cars0@chicago new_cars1@newark\
new_cars2@columbus
```

Line 1 specifies that the template name is **tem1** and the server to which to connect is the **detroit** server. Lines 2 and 3 list the names of the databases and database servers on which to realize the template.

The following example illustrates realizing the template on the source server, and then, creating the databases and tables, and loading data on the target database servers:

```
cdr realize template tem1 -c detroit\
--syncdatasource=detroit --extratargetrows=keep\
--foreground --memadjust=50M\
--target chicago newark columbus
```

```
cdr realize template tem1 -c detroit\
--syncdatasource=detroit --extratargetrows=keep\
--foreground --memadjust=50M\
--mode=receive_only chicago newark columbus
```

Line 1 realizes the template on the **detroit** server, as a primary server by default.

Line 2 specifies to use the **detroit** server as the source of the data to replicate to all other participating servers. If Enterprise Replication encounters any rows on the **chicago**, **newark**, or **columbus** servers that do not exist on the **detroit** server, those rows are kept.

Line 3 specifies that the synchronization operation is done in the foreground, and the size of the send queue is set to 50 MB.

Line 4 specifies the participant type for each server. The --target option makes all servers receive-only participants. The --mode=receive_only option makes each server a receive-only participant.

The following example verifies the database and table attributes on the **chicago**, **newark**, and **columbus** servers; the template is not realized on these servers:

```
cdr realize template tem1 -c detroit\
--verify chicago newark columbus
```

Related reference

cdr define template on page 367

cdr delete template on page 381

cdr list template on page 410

cdr define server on page 357

Related information

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr remaster

The cdr remaster command changes the SELECT clause or the server from which to base the master replicate definition of an existing master replicate. This command can also convert a classic (non-master) replicate to a master replicate.

Syntax

```
cdr remaster [ <Connect Option > (explicit id ) ]{ --master=server replicate [ modifier ] [ --erkey ] <Removing columns> }
```

Removing columns

```
    " --remove "

    " --database=database  --table=table column "

    "[ --wait= { seconds  |  -1 } ] "
```

Element	Purpose	Restrictions	Syntax
column	Name of the column to remove from replication.	The column must exist and be a replicated column.	
database	Name of the database from which to remove one or more columns from replication.	The database must exist and contain replicated tables.	
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier on page 257
replicate	Name of the replicate to be mastered.	The replicate must exist.	Long Identifiers on page 256
server	Name of the database server group from which to base the master replicate definition.	The name must be the database server group name.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
table	Name of the table from which to remove one or more columns from replication.	The table must exist and belong to a replicate.	

The following table describes the options to the cdr remaster command.

Long Form	Short Form	Meaning
--database=	-d	Specifies the database name from which to delete replicated columns.
--erkey	-K	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the participant definition, if the table that is being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used as the replication key.
--master=	-M	Specifies that the replicate being created is a master replicate.
--remove	-r	Removes the specified columns from replicate definitions.
--table=	-t	Specifies the table name from which to remove one or more replicated columns.
--wait=	-w	Specifies how long to wait for remastering to complete. Default is -1: wait indefinitely until all replicates are finished being remastered. If the remaster operation is not complete at the end of the waiting time, the operation is rolled back and the columns are not removed.

Usage

Remastering updates the replicate definition in the global catalogs of the replication servers. Use the cdr remaster command to perform one of the following tasks:

- Convert a classic replicate to a master replicate. Master replicates ensure schema consistency among the participants in the replicates.
- Update the definition of a master replicate whose participant was changed in an alter operation. You can change the SELECT clause or the server from which to base the master replicate definition.
- Remove one or more replicated columns from one or more replicates. The columns can belong to different replicates. You do not need to know the names of the replicates.

To use the cdr remaster command, the master replicate definition must be created with name verification turned on, by using the cdr define replicate command with the --name=y option.

Use the --erkey option if you are adding ERKEY columns to the participant definition, or if you are changing a participant definition that contains the ERKEY shadow columns.

You can run this command from within an SQL statement by using the SQL administration API.

The remastering operation creates temporary shadow replicates that are deleted when the remastering operation is complete. If shadow replicates exist, the remastering operation is in progress. You can run the cdr list replicate command to determine if the shadow replicate exists. An example of a shadow replicate name is:

```
Shadow_4_Repl1_GMT1090373046_GID10_PID28836
```

Shadow replicate names have the following format:

```
Shadow_4_basereplicatename_GMTtime_GIDgroupID_PIDpid
```

basereplicatename

The name of the replicate that is being remastered. If the replicate name is longer than 64 characters, only the first 64 characters are included.

time

The time stamp of when the shadow replicate was created, in GMT.

groupID

The group ID of the server. The group ID is the number that is specified by the -i option in the group definition in the `sqlhosts` file.

pid

The process ID of the client computer.

Example

Example: Add columns to a replicate definition

The following command shows the original definition of the master replicate before the alter operation:

```
cdr define repl --master=delhi -C timestamp\
newrepl "test@delhi.tab" "select col1, col2 from tab"\
```

The following command shows the cdr remaster command adding a column, **col3**, in the **newrepl** participant:

```
cdr remaster --master=delhi newrepl\
"select col1, col2, col3 from tab"
```

The following command shows adding the ERKEY shadow columns after the table was altered to include them:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3 from tab"
```

The following command shows changing the participant in the previous example to add another column and to continue to include the ERKEY shadow columns:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3, col4 from tab"
```

Example

Example: Remove columns from replicate definitions

The following command removes three columns from the database **mydb**: the column **prefix** from the table **customer** and the columns **discount** and **season** from the table **sales**:

```
cdr remaster --remove --database=mydb --table=customer prefix \
--table=sales discount season
```

The following command removes one column each from the databases **mydb1**, **mydb2**, and **mydb3**:

```
cdr remaster --remove --database=mydb1 --table=customer prefix \
--database=mydb2 --table=cars brand \
--database=mydb3 --table=regions northwest
```

Related reference

cdr alter on page 290

Related information

Remastering a Replicate on page 195

Removing replicated columns on page 191

Preparing tables without primary keys on page 79

Enterprise Replication Server administrator on page 18

cdr remaster gridtable

The cdr remaster gridtable command validates tables in a grid after an alter operation.

Syntax

```
cdr remaster gridtable
```

Usage

You can run the cdr remaster gridtable command to check whether tables in a grid have consistent metadata. The cdr remaster gridtable command checks every table in a grid on every grid server. The cdr remaster gridtable command is run automatically after a grid table is altered.

Return codes

A return code of 0 indicates that the command was successful.

Example

Examples

The following command checks all grid tables for consistency:

```
cdr remaster gridtable
```

Related reference

cdr change gridtable on page 297

Related information

Defining tables for grid queries on page 147

Grid queries on page 145

GRID clause on page

cdr remaster replicateset

The cdr remaster replicateset command updates the definitions of the set of replicates whose participants were changed by ALTER operations.

Syntax

```
cdr remaster replicateset [ <Connect Option > (explicit id ) ] --master=server derived_set [ --wait= { seconds | -1 } ]
```

Element	Purpose	Restrictions	Syntax
derived_set	Name of the derived replicate set to be mastered.	The derived replicate set must exist.	Long Identifiers on page 256
seconds	Number of seconds to wait for remastering to complete.	The number must be -1 or a positive integer.	
server	Name of the database server group from which to base the replicate definitions.	The name must be the database server group name.	Long Identifiers on page 256

The following table describes the options to the cdr remaster replicateset command.

Long Form	Short Form	Meaning
--master=	-M	Specifies the server from which to base the definitions of the replicates.
--wait=	-w	Specifies how long to wait for remastering to complete. Default is -1: wait indefinitely until all replicates are finished being remastered. If you specify a waiting time, but the remaster operation is not complete at the end of the waiting time, the operation is rolled back and the replicate definitions are not updated.

Usage

All participant servers in the derived replicate set must be online and the cdr utility must be able to connect to each participant.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

When you change replicate participants by running ALTER operations, you must remaster the replicate. Remastering updates the replicate definition in the global catalogs of the replication servers. Before you can run the cdr remaster replicateset command, you must run the cdr define replicateset command with the --needRemaster option to create a derived replicate set.

You can run this command from within an SQL statement by using the SQL administration API.

The remastering operation creates temporary shadow replicates that are deleted when the remastering operation is complete. If shadow replicates exist, the remastering operation is in progress. You can run the cdr list replicate command to determine if the shadow replicate exists. An example of a shadow replicate name is:

```
Shadow_4_Repl1_GMT1090373046_GID10_PID28836
```

Shadow replicate names have the following format:

```
Shadow_4_basereplicatename_GMTtime_GIDgroupID_PIDpid
```

basereplicatename

The name of the replicate that is being remastered. If the replicate name is longer than 64 characters, only the first 64 characters are included.

time

The time stamp of when the shadow replicate was created, in GMT.

groupID

The group ID of the server. The group ID is the number that is specified by the -i option in the group definition in the `sqlhosts` file.

pid

The process ID of the client computer.

Example

Example

The following command remasters a derived replicate set named **derived_accounts** and sets the replication server named **server1** as the master server:

```
cdr remaster replicateset --master=server1 derived_accounts
```

Related information

Altering multiple tables in a replicate set on page 190

cdr remove onconfig

The **cdr remove onconfig** command removes the specified value from a configuration parameter in the ONCONFIG file.

Syntax

```
cdr remove onconfig " parameter namevalue "
```

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter from which to remove the value.	Not all configuration parameters can be changed with this command. Only the following parameters can be changed: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPRESS_ATSRISWARN • ENCRYPT_CIPHERS • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: ◦ CDRSITES_731 ◦ CDRSITES_92X ◦ CDRSITES_10X	
value	The value of the configuration parameter to remove.	Must be an existing value of the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr remove onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

Suppose the ENCRYPT_MAC configuration parameter is set to allow medium and high encryption levels, so that it appears in the ONCONFIG file as: `ENCRYPT_MAC medium,high`. The following command removes the medium encryption level and retains only the high encryption level:

```
cdr remove onconfig "ENCRYPT_MAC medium"
```

Suppose the CDR_SITES_92X environment variable specifies the cdrIDs of 3, 4, and 5, so that it appears in the ONCONFIG file as: CDR_ENV CDR_SITES_92X=3,4,5. The following command removes the cdrID of 3 from the list of supported version 9.2x servers:

```
cdr remove onconfig "CDR_ENV CDR_SITES_92X=3"
```

Related reference

cdr add onconfig on page 289

cdr change onconfig on page 298

Related information

Dynamically Modifying Configuration Parameters for a Replication Server on page 162

Enterprise Replication Server administrator on page 18

cdr repair

The **cdr repair** command synchronizes data based on ATS or RIS files.

Syntax

```
cdr repair [ { --check | { --verbose | --quiet } } ] { atsats_file | risris_file }
```

Element	Purpose	Restrictions	Syntax
ats_file	Name of the file for Aborted Transaction Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 bytes. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.
ris_file	Name of the file for Row Information Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 characters. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.

The following table describes the option to **cdr repair**.

Long Form	Short Form	Meaning
--check	**-C**	Check the consistency between the database server and the ATS or RIS file. Display repair operations to stderr, but do not perform the repair operations. In an active system, operations displayed with this option will not necessarily match those performed later during an actual repair.
--quiet	**-q**	Quiet mode. Repair operations are not displayed to stderr.
--verbose	**-v**	Verbose mode (default). All repair operations are displayed to stderr.

Usage

The **cdr repair** command reconciles rows that failed to be applied based on the information in the specified ATS or RIS file. If a row exists on the source database server, it is replicated again. If a row does not exist on the source database server, but does exist on the target server, then it is deleted from the target database server. By default, each of the repair operations is displayed to stderr.

If you are running this command as a DBSA, you must have read permission on the ATS and RIS files. Permissions on ATS and RIS files can be set with the **chown** operating system command.

The ATS or RIS file you specify in the **cdr repair** command must be in text format, which is the default format. You cannot specify the XML format of an ATS or RIS in the **cdr repair** command.

Before you run a repair, preview the repair to make sure the operations that would be performed are correct. To preview the repair operations, use the **-check** option. All repair operations are displayed to stderr, but not performed. In an active system, however, the operations displayed by the **-check** option might not be the same as the operations performed when you later run the repair.

The server on which you run the **cdr repair** command must have a copy of the ATS or RIS file and be able to connect to the source and target database servers involved in the failed transaction. In a hierarchical routing environment where the source and target database servers are not directly connected you might need to run the **cdr repair** command from an intermediate server. If necessary, copy the ATS or RIS file to the intermediate server.

ATS and RIS files do not include code set information, therefore, the code sets associated with the locales specified by the DB_LOCALE and CLIENT_LOCALE environment variables must be the same.

You can run this command from within an SQL statement by using the SQL administration API.

Note: The cdr repair command is not supported for replicates that are defined with the `--UTF8=y` option. For replicates that are defined with the `--UTF8=y` option, use the cdr check replicate --repair or cdr check replicateset --repair command to repair data.

Example

Examples

The following example repairs inconsistencies between the **g_beijing** and **g_amsterdam** servers resulting from an aborted transaction:

```
% cdr repair ats ats.g_beijing.g_amsterdam.D_2.070827_12:58:55.1
Attempting connection to syscdr@amsterdam...
Using syscdr@amsterdam.
Source ID:10 / Name:g_amsterdam
Target ID:20 / Name:g_beijing
(1) [s = "1"]: Row will be updated on the source for replicate <655361>
(2) [s = "2"]: Row not found on the source for replicate <655361>
(2) [s = "2"]: Row not found on the target for replicate <655361>
(2) [s = "2"]: No operation will be performed.
(3) [s = "3"]: Row will be updated on the source for replicate <655361>
(4) [s = "4"]: Row will be updated on the source for replicate <655361>
(5) [s = "5"]: Row will be updated on the source for replicate <655361>
(6) [s8 = "1911"]: Row will be updated on the source for replicate <655362>
(7) [s8 = "1912"]: Row will be updated on the source for replicate <655362>
(8) [s8 = "1913"]: Row will be updated on the source for replicate <655362>
(9) [s8 = "1914"]: Row will be updated on the source for replicate <655362>
(10) [s8 = "1915"]: Row will be updated on the source for replicate <655362>
```

Related reference

cdr view on page 486

Related information

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr reset qod

The cdr reset qod command resets failed-transaction counts for replicates on replicate servers. Connection Manager service-level agreements (SLA) that contains a FAILURE or LATENCY redirection policy use failed-transaction counts to determine where to route client requests.

Syntax

cdr reset qod [<Connect Option > (explicit id)] { **--repl**=*replicate_name* | **--replset**=*repl_set_name* | **--allrepl** } [*server_name*] [**--verbose**]

Element	Purpose	Restrictions	Syntax
replicate_name	The name of the replicate.	The replicate must exist.	Long Identifiers on page 256
repl_set_name	The name of the replicate set.	The replicate set must exist.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
server_name	The name of the server.	Must be the name of an existing database server group in SQLHOSTS. Cannot be a leaf server.	Long Identifiers on page 256

The following table describes the options to the cdr reset qod command.

Long Form	Short Form	Meaning
--allrepl	-A	Resets the failed-transaction count on all replicates.
--repl=	-r	Specifies the replicate for which to reset the failed-transaction count.
--replset=	-s	Specifies the replicate set for which to reset the failed-transaction count.
--verbose	-v	Displays details of the operations the command is performing

Usage

Use the cdr reset qod command to reset the failed-transaction count to zero for replicates or replicate sets on specified replication servers. Run the cdr reset qod command before you repair inconsistent data, so that you can count failures that occur after the repair.

You must run the cdr reset qod command from a non-leaf server. If you do not specify any servers to reset, the current server to which you are connected is reset. If you specify one or more servers to reset, you must explicitly include the server to which you are connected if you want to reset it.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 44, 217.

For information on error codes, see Return Codes for the cdr Utility on page 262.

Example

Example 1: Resetting failed-transaction counts for a specific replicate on a specific replication server

The following example resets the failed-transaction count for **replicate_1** on **server_1**:

```
cdr reset qod --repl=replicate_1 server_1
```

Example

Example 2: Resetting failed-transaction counts for all replicates on specific replication servers

The following example resets the failed-transaction count for all replicates on **server_2** and **server_3**:

```
cdr reset qod --allrepl server_2 server_3
```

Example

Example 3: Resetting failed-transaction counts for all replicates in a specific replicate set on a specific replication server

The following example resets the failed-transaction count for all replicates in **replicate_set_1** on **server_4**.

```
cdr reset qod --replset=replicate_set_1 server_4
```

Example

Example 4: Resetting failed-transaction counts for all replicates in specific replicate sets on a specific replication server

The following example resets the failed-transaction count for all replicates in **replicate_set_2** and **replicate_set_4** on **server_5**.

```
cdr reset qod -s replicate_set_2 -s replicate_set_4 server_5
```

Example

Example 5: Resetting failed-transaction counts for all replicates on a specific replication server, and displaying operation details

The following example connects to **server_6**, and then resets the failed-transaction count for all of replicates on **server_6**. The command displays details of the operations that are performed:

```
cdr reset qod -c -A server_6  -v
```

Figure 24. Output of cdr reset qod with verbose details.

```
Resetting Quality of Data on server_6
  Resetting replicate replicate_1
  Resetting replicate replicate_2
  Resetting replicate replicate_3
  Resetting replicate replicate_4
  Resetting replicate replicate_5
  Resetting replicate replicate_6
```

Related reference

cdr define qod on page 339

cdr start qod on page 444

cdr stop qod on page 467

Related information

SLA Connection Manager configuration parameter on page

Enterprise Replication Server administrator on page 18

cdr resume replicate

The **cdr resume replicate** command resumes delivery of replication data.

Syntax

```
cdr resume replicate [ <Connect Option > (explicit id ) ] repl_name
```

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to change to active state.	The replicate must be suspended.	Long Identifiers on page 256

Usage

The **cdr resume replicate** command causes all participants in the replicate *repl_name* to enter the active state.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr resume replicate** to resume that individual replicate. You must use **cdr resume replicateset** to resume all replicates in the exclusive replicate set. If a replicate belongs to a non-exclusive replicate set, you can resume the individual replicates in the set.

When you run the **cdr resume replicate** command, an event alarm with a class ID of 57 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate **smile**:

```
cdr res repl smile
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr start replicate on page 445

cdr stop replicate on page 468

cdr suspend replicate on page 471

Enterprise Replication Event Alarms on page 221 cdr list server on page 402

cdr resume replicateset on page 441

Related information

Exclusive Replicate Sets on page 118

Enterprise Replication Server administrator on page 18

cdr resume replicateset

The **cdr resume replicateset** command resumes delivery of replication data for all the replicates in a replicate set.

Syntax

```
cdr resume replicateset [ <Connect Option > (explicit id ) ] repl_set
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to resume.	None.	Long Identifiers on page 256

Usage

The **cdr resume replicateset** command causes all replicates contained in the replicate set *repl_set* to enter the active state for all participants.

If not all the replicates in a non-exclusive replicate set are suspended, the **cdr resume replicateset** command displays a warning and only resumes the replicates that are currently suspended.

When you run the **cdr resume replicateset** command, an event alarm with a class ID of 58 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate set **accounts_set**:

```
cdr res replset accounts_set
```

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr define replicate on page 342

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

Enterprise Replication Event Alarms on page 221

cdr resume replicate on page 440

cdr list server on page 402

Related information

Enterprise Replication Server administrator on page 18

cdr resume server

The **cdr resume server** command resumes delivery of replication data to a suspended database server.

Syntax

```
cdr resume server [ <Connect Option > (explicit id ) ] to_server_group [ from_server_group ]
```

Element	Purpose	Restrictions
to_server_group	Name of the database server group to which to resume delivery of replication data.	The database server group must be currently active in Enterprise Replication.
from_server_group	Name of the database server group from which to resume sending data to *to_server_group*.	The database server group must be currently active in Enterprise Replication.

Usage

The **cdr resume server** command resumes delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command resumes replication of data from all database servers participating in the Enterprise Replication system to the *to_server_group*. Replication data must have previously been suspended to the server with the **cdr suspend server** command.

When you run the **cdr resume server** command, an event alarm with a class ID of 52 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) and resumes replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

```
cdr res serv g_iowa g_ohio g_utah
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr delete server on page 377

cdr disconnect server on page 385

cdr list server on page 402

cdr modify server on page 421

cdr suspend server on page 473

Enterprise Replication Event Alarms on page 221

Related information

Enterprise Replication Server administrator on page 18

cdr start

The **cdr start** command starts Enterprise Replication processing.

Syntax

```
cdr start [ <Connect Option >(explicit id ) ]
```

Usage

Use **cdr start** to restart Enterprise Replication after you stop it with the **cdr stop** command or replication stops for another reason, such as memory allocation problems. When you issue **cdr start**, Enterprise Replication activates all connections to other connected replication servers. Replication servers, replicates, and replicate sets that were suspended before the **cdr stop** command was issued remain suspended; no data is sent for the suspended servers, replicates, or sets.

Enterprise Replication resumes evaluation of the logical log (if required for the instance of Enterprise Replication) at the *replay* position. The replay position is the position where Enterprise Replication stops evaluating the logical log when replication is stopped. When replication resumes, all appropriate database transactions that occurred while replication was stopped are replicated. If replication is stopped for a prolonged period of time, the replay position in the logical log might be overwritten. If the replay position is not available, the **cdr start** command fails with return code 214 and event alarm 75 is raised. In this situation, you must empty the send queues and reset the replay position by running the **cdr cleanstart** command, and then synchronize the data.

When you run the **cdr start** command, event alarm 49 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Important: Whenever replication is stopped, data can become inconsistent. Therefore, issue **cdr start** and **cdr stop** with extreme caution.

Example

Examples

The following example restarts Enterprise Replication processing on database server **utah**:

```
cdr start -c utah
```

Related reference

cdr cleanstart on page 337

cdr list server on page 402

cdr stop on page 465

Enterprise Replication Event Alarms on page 221

Related information

Restarting Replication on a Server on page 165

Enterprise Replication Server administrator on page 18

cdr start qod

The cdr start qod command starts quality of data (QOD) monitoring for replication servers.

Syntax

```
cdr start qod [ <Connect Option > (explicit id ) ]
```

Usage

Use the cdr start qod command to start monitoring the quality of data for replications servers. If Connection Manager service-level agreements (SLAs) use a apply-failure or transaction-latency redirection policy, the Connection Manager uses quality of data information to decide where to route client connections.

Quality of data information is used for the following SLA redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.

You must run the cdr define qod command to define a master server before you can run the cdr start qod command. The cdr start qod command must be run on the master server.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see Return Codes for the cdr Utility on page 262.

Example

Example

The following command starts quality of data monitoring when it is run on the master server:

```
cdr start qod
```

Related reference

cdr define qod on page 339

cdr stop qod on page 467

cdr reset qod on page 437

Related information

SLA Connection Manager configuration parameter on page

Enterprise Replication Server administrator on page 18

cdr start replicate

The **cdr start replicate** command starts the capture and transmittal of replication transactions.

Syntax

cdr start replicate [<Connect Option > (explicit id)] *repl_name* [*server_group*] [**--syncdatasource**=*data_server* <Synchronization Options>]

Synchronization Options

“[**--extratargetrows**={ **delete** | **keep** | **merge** }]”

“[**--foreground** [**--memadjust**= *size* { **K** | **M** }]]”

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
repl_name	Name of the replicate to start.	The replicate must exist.	Long Identifiers on page 256
server_group	Name of database server groups on which to start the replicate.	The database server must be in an Enterprise Replication domain.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicate** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicate** command causes the replicate to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the --syncdatasource option as a DBSA, you must have certain permissions granted to you on several system tables in the syscdr database. For more information, see Preparing for Role Separation (UNIX) on page 80.

If you would like the synchronization operation to be run in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If no server is specified, the *repl_name* starts on all servers that are included in the replicate. A replicate can have both active and inactive participants. When at least one participant is active, the replicate is active, however, replication does not start until at least two participants are active. You cannot start replicates that have no participants.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr start replicate** to start that individual replicate. You must use **cdr start replicateset** to start all replicates in the exclusive replicate set.

Because Enterprise Replication does not process log records that were produced before the **cdr start replicate** command was run, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.

You can run the cdr check queue --qname=cntrlq command to wait for the **cdr start replicate** command to be applied at all Enterprise Replication servers before running the data synchronization task.

When you run the **cdr start replicate** command, an event alarm with a class ID of 59 is generated, if that event alarm is enabled.

Example

Examples

The following command starts the replicate **accounts** on the server groups **g_svr1** and **g_svr2**:

```
cdr sta rep accounts g_svr1 g_svr2
```

The following example starts the replicate named **accounts** on the server **g_svr1** with **g_svr2** as the source server:

```
cdr start replicate accounts g_svr1 --syncdatasource=g_svr2\
--foreground --memadjust=50M
```

The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr stop replicate on page 468

cdr suspend replicate on page 471

Enterprise Replication Event Alarms on page 221

Related information

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr start replicateset

The **cdr start replicateset** command starts the capture and transmittal of replication transactions for all the replicates in a replicate set.

Syntax

```
cdr start replicateset [ <Connect Option > (explicit id ) ] repl_set [ server_group ] [  --syncdatasource=data_server
<Synchronization Options> ]
```

Synchronization Options

```
“[  --extratargetrows={delete  | keep  | merge}]”

“[  --foreground [  --memadjust= size {K  | M}]]”
```

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be defined in Enterprise Replication.	
repl_set	Name of replicate set to start.	The replicate set must exist.	Long Identifiers on page 256
server_group	Names of database server groups on which to start the replicate set.	The database server groups must be defined for Enterprise Replication.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicateset** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicateset** command causes the replicates defined in the specified replicate set to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the --syncdatasource option as a DBSA, you must have certain permissions granted to you on several system tables in the syscdr database. For more information, see Preparing for Role Separation (UNIX) on page 80.

If you would like the synchronization operation to be run as in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If the *server_group* list is omitted, the replicate set *repl_set* enters the active state for all database servers participating in the replicate set.

Because Enterprise Replication does not process log records that were produced before the **cdr start replicateset** command took place, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicateset** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicates in the replicate set.

If not all the replicates in a non-exclusive replicate set are inactive, the **cdr start replicateset** command displays a warning and only starts the replicates that are currently inactive.

When you run the **cdr start replicateset** command, an event alarm with a class ID of 60 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default database server specified by the **INFORMIXSERVER** environment variable and starts the replicate set **accounts_set** on the server groups **g_hill** and **g_lake**:

```
cdr sta replset accounts_set g_hill g_lake
```

The following example starts the replicate set **accounts_set** on the server **g_hill** with **g_lake** as the source server:

```
cdr start replicateset accounts_set g_hill --syncdatasource=g_lake\
--foreground --memadjust=50M
```

The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr define replicate on page 342

cdr stop replicateset on page 469

cdr suspend replicateset on page 472

Enterprise Replication Event Alarms on page 221

Related information

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr start sec2er

The cdr start sec2er command converts a high availability cluster to replication servers.

Syntax

```
cdr start sec2er [ <Connect Option > (explicit id ) ] secondary
```

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		Long Identifiers on page 256

Usage

You must run the cdr start sec2er command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The cdr start sec2er command converts the two cluster servers into replication servers.

The following conditions must be met on both the primary and secondary cluster servers before running the cdr start sec2er command:

- The `sqlhosts` files must be configured for Enterprise Replication:
  - Each server must belong to a different group.
  - The group identifier for each server must be different.
  - The `sqlhosts` files on each server must contain a server and a group entry for the other server.
- All databases and tables must be logged.
- No tables can be defined with label-based access control.
- Typed tables must have primary keys.
- User-defined types must support Enterprise Replication.
- The CDR_QDATA_SBSPACE configuration parameter must be set.
- Both server must be running Informix® version 11.10 or later.
- If the servers are running Informix® database software prior to 11.70, Enterprise Replication cannot be defined.
- Enterprise Replication must be active if it is already defined on either of the servers.

The cdr start sec2er command performs the following tasks:

- The servers are defined as replication servers.
- Any tables on the primary server that do not have a primary key are altered to add ERKEY shadow columns.
- A replicate is created and started for each user table on the primary server.

If the cdr start sec2er command fails or is interrupted, you might see the following error message:

```
ERROR:  Command cannot be run on pre-11.70 instance if ER is already running.
```

If you receive this error, remove replication by running the cdr delete server command for both servers and then run the cdr start sec2er command again.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error codes is returned: 225.

For information on these error codes, see Return Codes for the cdr Utility on page 262.

Example

Example

The following example converts a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** into replication servers. The output of the cdr start sec2er command shows the commands that are run.

```
$cdr start sec2er secserv
--
--  Define ER for the first time
--
cdr define serv -c priserv -I priserv

--
--  Creating Replication Key
--
dbaccess - - <<EOF
database stores_demo;
alter table 'bill'.classes add ERKEY;
EOF

--
--  Define the replicates
--
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=priserv sec2er_1_1282611664_call_type --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.call_type" \
         "select * from 'bill'.call_type"
cdr start repl --connect=priserv sec2er_1_1282611664_call_type

cdr define repl --connect=priserv sec2er_2_1282611664_cust_calls --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.cust_calls" \
         "select * from 'bill'.cust_calls"
cdr start repl --connect=priserv sec2er_2_1282611664_cust_calls

. . .

cdr define repl --connect=priserv sec2er_5_1282611664_customer --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.customer" \
         "select * from 'bill'.customer"
cdr start repl --connect=priserv sec2er_5_1282611664_customer

cdr define repl --connect=priserv sec2er_6_1282611664_classes --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.classes" \
         "select * from 'bill'.classes"
cdr start repl --connect=priserv sec2er_6_1282611664_classes
```

```
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related reference

CDR_QDATA_SBSPACE Configuration Parameter on page 512

cdr check sec2er on page 334

Example of creating a new replication domain by cloning on page 95

Related information

Configuring ports and service names for replication servers on page 57

Preparing Logging Databases on page 80

Enterprise Replication Server administrator on page 18

cdr stats rqm

The **cdr stats rqm** command displays information about the reliable queue manager (RQM) queues used for Enterprise Replication.

Syntax

```
cdr stats rqm [ <Connect Option > (explicit id ) ] [ { --ackq | --cntrlq | --recvq | --syncq | --sendq } ]
```

The following table describes the **cdr stats rqm** options.

Long Form	Short Form	Meaning
--ackq	**-A**	Prints the statistics for the acknowledgment send queue.
--cntrlq	**-C**	Prints the statistics for the control send queue.
--recvq	**-R**	Prints the statistics for the receive queue.
--syncq	**-S**	Prints the statistics for the sync send queue.
--sendq	**-T**	Prints the statistics for the send queue.

Usage

The **cdr stats rqm** command displays the RQM (reliable queue manager) statistics for the queues used by Enterprise Replication. These queues are the ack send, control send, send, sync send, and the receive queue. If no queue is specified, the **cdr stats rqm** command displays statistics for all Enterprise Replication queues.

The **cdr stats rqm** command shows, among other things, how many transactions are currently queued in memory and spooled, the size of the data in the queue, how much real memory is being used, pending transaction buffers and data, the maximum memory used for data and headers (overhead), and totals for the number of transactions queued, the number of transactions, the number of deleted transactions, and the number of transaction lookups that have occurred.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieves the statistics for its Enterprise Replication queues.

Example

Examples

The following example shows the output for **cdr stats rqm --ackq**:

```
RQM Statistics for Queue number: 1 name:  ack_send
 Flags:                     ACKSEND_Q, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 44 Bytes
 Max Real memory hdrs used: 320 Bytes
 Total data queued:         120 Bytes
 Total Txns queued:         0
 Total Txns                 3
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        3
 Total Txns duplicated:     0
 Total Txn Lookups:         8
```

The following example shows the output for **cdr stats rqm --cntrlq**:

```
RQM Statistics for Queue number: 2 name:  control_send
 Transaction Spool Name:    control_send_stxn
 Flags:                     CTRL_SEND_Q, STABLE, USERTXN, PROGRESS_TABLE,
                            NEED_ACK, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 185 Bytes
 Max Real memory hdrs used: 320 Bytes
 Total data queued:         185 Bytes
 Total Txns queued:         0
 Total Txns                 1
 Total Txns spooled:        1
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        1
 Total Txns duplicated:     0
 Total Txn Lookups:         4
```

The following example shows the output for **cdr stats rqm --recvq**:

```
RQM Statistics for Queue number: 4 name:  trg_receive
 Transaction Spool Name:    trg_receive_stxn
 Flags:                     RECV_Q, SPOOLED, PROGRESS_TABLE
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 0 Bytes
 Max Real memory hdrs used: 0 Bytes
 Total data queued:         0 Bytes
 Total Txns queued:         0
 Total Txns                 0
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        0
 Total Txns duplicated:     0
 Total Txn Lookups:         0
```

The following example shows the output for **cdr stats rqm --syncq**:

```
RQM Statistics for Queue number: 3 name:  sync_send
 Flags:                     SYNC_Q, NEED_ACK, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 0 Bytes
 Max Real memory hdrs used: 0 Bytes
 Total data queued:         0 Bytes
 Total Txns queued:         0
 Total Txns                 0
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        0
 Total Txns duplicated:     0
 Total Txn Lookups:         1131
```

The following example shows the output for **cdr stats rqm --sendq**:

```
RQM Statistics for Queue number: 0 name:  trg_send
 Transaction Spool Name:    trg_send_stxn
 Flags:                     SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK,
                            SENDQ_MASK, SREP_TABLE
 Txns in queue:             12
 Txns in memory:            12
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           24960
 Size of Data in queue:     24960 Bytes
 Real memory in use:        24960 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 24960 Bytes
 Max Real memory hdrs used: 22080 Bytes
 Total data queued:         27560 Bytes
 Total Txns queued:         0
 Total Txns                 14
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        2
 Total Txns duplicated:     0
 Total Txn Lookups:         28
```

Related information

Enterprise Replication Server administrator on page 18

cdr stats recv

The **cdr stats recv** command displays receiver parallelism statistics and latency statistics by source node.

Syntax

```
cdr stats recv [ <Connect Option > (explicit id ) ]
```

Usage

The **cdr stats recv** command displays the parallelism statistics for the receiver, including transaction count, number of pending and active transactions, the maximum that have been pending and active, the average number of pending and active transactions, and the commit rate. Totals and averages are calculated for pending and active transactions for the servers listed.

The Statistics by Source report shows the breakdown of transactions (number of inserts, updates, and deletes) and the latest source commit time and target apply time per server. The replication latency is the difference between the time when the transaction was committed on the source server and the time when the same transaction is applied on the target.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieved the statistics from it.

Example

Examples

The following output is an example of the **cdr stats recv** command:

```
cdr stats recv

Receive Parallelism Statistics
Server Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
   144     11         0      0      3      2    1.27    1.36    0.01

Tot Pending:0       Tot Active:0    Avg Pending:0.00    Avg Active:0.00

Avg Commit Rate:0.01


Statistics by Source

Server  Repl     Txn   Ins   Del   Upd   Last Target Apply    Last Source Commit
144     9371650   11     0     0   220   2005/03/30 09:36:25  2005/03/30 09:36:25
```

Related information

Enterprise Replication Server administrator on page 18

cdr stats check

The **cdr stats check** command displays the progress of a consistency check that specified a progress report task name.

cdr stats check [<Connect Option > (explicit id)] [**--repeat**=*time*] [**--verbose**] [**--delete**=*task_name*] *task_name*

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	Long Identifiers on page 256
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats check** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats check** command to display the progress of a consistency check operation while the **cdr check replicate** or **cdr check replicateset** command is running. You must have specified a task name in the **cdr check replicate** or **cdr check replicateset** command. You must be connected to the same server on which the **cdr check replicate** or **cdr check replicateset** command was run when you run the **cdr stats check** command.

The **cdr stats check** command displays a snapshot of the consistency report and an estimate of the time remaining to complete the consistency check. If you use the **--repeat** option, the consistency report is displayed every specified time interval.

You can view the progress of previously run consistency checks that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Example

Examples

The following example checks a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr check repl -r repl1 -m cdr1 -a --name=tst
cdr stats check --repeat=2 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:10:59
*********+----+----+----+----+----+----+----+----+ Remaining 0:00:08

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
**********************--+----+----+----+----+----+ Remaining 0:00:04
```

```
---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
***********************************----+----+----+ Remaining 0:00:02

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
*************************************************+ Remaining 0:00:01

---------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example checks and repairs the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr check repl -r repl1 -m cdr1 -a --name=tst --repair
cdr stats check --repeat=4 --verbose tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:34:42
*******--+----+----+----+----+----+----+----+----+ Remaining 0:00:12

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                  9000         0         0         0         0         0
cdr2                  9000         0         0        99         0        99
cdr3                  9000         0         0         0         0         0

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
*********************************-+----+----+----+ Remaining 0:00:02

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 43000         0         0         0         0         0
cdr2                 43000         0         0        99         0        99
cdr3                 43000         0         0         0         0         0

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
*************************************************+ Remaining 0:00:01

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 39000         0         0         0         0        99
cdr2                 38901         0        99        99         0        99
cdr3                 39000         0         0         0         0         0

---------------------------------------------------
```

```
Job tst
repl1                    Completed
                         Started Jan 17 16:34:42, Elapsed Time 0:00:11

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 64099         0         0         0         0        99
cdr2                 64000         0        99        99         0        99
cdr3                 64099         0         0         0         0         0
```

The following example checks a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr check replset -s set -m cdr1 -a -n tst
cdr stats check -r 5 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl3                    Started Jan 17 16:41:19
*****----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:52

---------------------------------------------------
Job tst
repl3                    Started Jan 17 16:41:19
***************************************+----+----+ Remaining 0:00:01
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:19

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Started Jan 17 16:41:27
*******************+----+----+----+----+----+----+ Remaining 0:00:06
repl1                    Pending
Estimated time remaining for job tst  0:00:13

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Started Jan 17 16:41:35
----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
Estimated time remaining for job tst  0:01:08

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
```

```
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Started Jan 17 16:41:35
*********************************-+----+----+----+ Remaining 0:00:02
Estimated time remaining for job tst  0:00:02

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Completed
                         Started Jan 17 16:41:35, Elapsed Time 0:00:11
Run time for job tst  0:00:27
```

Related reference

The replcheck_stat Table on page 572

The replcheck_stat_node Table on page 573

cdr check replicate on page 313

cdr check replicateset on page 325

Related information

Checking Consistency and Repairing Inconsistent Rows on page 180

Enterprise Replication Server administrator on page 18

cdr stats sync

The **cdr stats sync** command displays the progress of a synchronization operation that specified a progress report task name.

cdr stats sync [<Connect Option > (explicit id)] [**--repeat=***time*] [**--verbose**] [**--delete=***task_name*] *task_name*

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	Long Identifiers on page 256
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats sync** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.

Long Form	Short Form	Meaning
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats sync** command to display the progress of a synchronization operation (**cdr sync replicate** or **cdr sync replicateset**). You must be connected to the same server on which the **cdr sync replicate** or **cdr sync replicateset** command was run when you run the **cdr stats sync** command. The **cdr stats sync** command displays a snapshot of the progress report and an estimate of the time remaining to complete the synchronization operation. If you use the **--repeat** option, the progress report is displayed every specified time interval.

You can view the progress of previously run synchronization operations that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Example

Examples

The following example synchronizes a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr sync repl -r repl1 -m cdr1 -a --name=tst
cdr stats sync --repeat=2 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:10:59
*********+----+----+----+----+----+----+----+----+ Remaining 0:00:08

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
**********************--+----+----+----+----+----+ Remaining 0:00:04

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
***********************************----+----+----+ Remaining 0:00:02

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
```

```
*************************************************+ Remaining 0:00:01

---------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example synchronizes the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr sync repl -r repl1 -m cdr1 -a --name=tst
cdr stats sync --repeat=4 --verbose tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:34:42
*******--+----+----+----+----+----+----+----+----+ Remaining 0:00:12

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                  9000         0         0         0         0         0
cdr2                  9000         0         0        99         0        99
cdr3                  9000         0         0         0         0         0

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
*********************************-+----+----+----+ Remaining 0:00:02

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 43000         0         0         0         0         0
cdr2                 43000         0         0        99         0        99
cdr3                 43000         0         0         0         0         0

---------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
*************************************************+ Remaining 0:00:01

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 39000         0         0         0         0        99
cdr2                 38901         0        99        99         0        99
cdr3                 39000         0         0         0         0         0

---------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:34:42, Elapsed Time 0:00:11

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 64099         0         0         0         0        99
cdr2                 64000         0        99        99         0        99
```

```
cdr3                   64099     0          0          0          0          0
```

The following example synchronizes a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr sync replset -s set -m cdr1 -a -n tst
cdr stats sync -r 5 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl3                    Started Jan 17 16:41:19
*****----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:52

---------------------------------------------------
Job tst
repl3                    Started Jan 17 16:41:19
***************************************+----+----+ Remaining 0:00:01
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:19

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Started Jan 17 16:41:27
*******************+----+----+----+----+----+----+ Remaining 0:00:06
repl1                    Pending
Estimated time remaining for job tst  0:00:13

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Started Jan 17 16:41:35
----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
Estimated time remaining for job tst  0:01:08

---------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Started Jan 17 16:41:35
*********************************-+----+----+----+ Remaining 0:00:02
Estimated time remaining for job tst  0:00:02

---------------------------------------------------
Job tst
repl3                    Completed
```

```
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Completed
                         Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                    Completed
                         Started Jan 17 16:41:35, Elapsed Time 0:00:11
Run time for job tst  0:00:27
```

Related reference

The replcheck_stat Table on page 572

The replcheck_stat_node Table on page 573

cdr sync replicate on page 476

cdr sync replicateset on page 481

Related information

Performing Direct Synchronization on page 178

Enterprise Replication Server administrator on page 18

cdr stop

The **cdr stop** command stops replication on the server to which you are connected without shutting down the database server.

Syntax

```
cdr stop <Connect Option> (explicit id )
```

Usage

Generally, to stop replication on a server, you should shut down the database server. Under rare conditions, you might want to temporarily stop the Enterprise Replication processing without shutting down the database server.

The **cdr stop** command shuts down replication in an orderly manner; however no data to be replicated is captured. When the shutdown of Enterprise Replication is complete, the message `CDR shutdown complete` appears in the database server log file.

Stopping replication has the following effects:

- There is no connection between the stopped server and active replication servers.
- Transactions on the stopped server are not captured for replication. However, after restarting replication, transaction capture restarts at the replay position. If replication is stopped for long enough that the replay position is overwritten, you must restart replication with the cdr cleanstart command. If the CDR_LOG_LAG_ACTION configuration parameter is set to logstage, logs are staged to protect the replay position.
- Transactions on active replication servers are queued for the stopped server, but there is the possibility of filling up the send queues.

- Control messages on active replication servers are queued for the stopped server.
- The only Enterprise Replication commands you can run on the stopped server are cdr start, cdr cleanstart, and cdr delete server with the --force option.

To ensure consistency, prevent database update activity while Enterprise Replication is stopped. Replication threads remain stopped until you issue a **cdr start** command. Shutting down and restarting the stopped database server does not restart replication.

If you plan to stop replication for a long period of time and your replicates use time stamp or delete wins conflict resolution rules, consider using the cdr disable server command instead of the cdr stop command.

When you run the **cdr stop** command, event alarms with class IDs of 50 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Return Codes

0

The command was successful.

5

Enterprise Replication cannot connect to the specified server.

48

There is not enough memory to perform the operation.

62

Enterprise Replication is not active.

93

Enterprise Replication is in the process of starting.

94

Enterprise Replication is already in the process of stopping.

Example

Examples

The following example stops Enterprise Replication processing on database server **paris**. Processing does not resume until a **cdr start** command restarts it:

```
cdr stop −c paris
```

Related reference

cdr start on page 443

Enterprise Replication Event Alarms on page 221

Related information

Resynchronizing Data among Replication Servers on page 177

Temporarily stopping replication on a server on page 164

Enterprise Replication Server administrator on page 18

cdr stop qod

The cdr stop qod command stops quality of data (QOD) monitoring for replication servers.

Syntax

```
cdr stop qod [ <Connect Option > (explicit id ) ]
```

Usage

Use the cdr stop qod command to stop monitoring quality of data for the replication servers.

The cdr stop qod command must be run on the master server defined by the cdr define qod command.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 217.

For information on this error code, see Return Codes for the cdr Utility on page 262.

Example

Example 1: Stopping quality of data monitoring from a master server

The following example stops quality of data monitoring:

```
cdr stop qod
```

This command must be run on the master server that was defined by the cdr define qod command.

Example

Example 2: Connecting to a master server, and then stopping quality of data monitoring

For the following example, **server_1** was defined as the master server by the cdr define qod command. The following example connects to **server_1**, and then stops quality of data monitoring:

```
cdr stop qod -c server_1
```

Related reference

cdr define qod on page 339

cdr start qod on page 444

cdr reset qod on page 437

Related information

SLA Connection Manager configuration parameter on page

Enterprise Replication Server administrator on page 18

cdr stop replicate

The **cdr stop replicate** command stops the capture, transmittal, and reception of transactions for replication.

Syntax

cdr stop replicate [<Connect Option > (explicit id)] *repl_name* [*at_server_group*]

Element	Purpose	Restrictions	Syntax
repl_name	Name of the new replicate.	The replicate must be active and not in an exclusive replicate set.	Long Identifiers on page 256
at_server_group	List of database server groups on which to stop the replicate.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicate** command changes the state of the replicate *repl_name* to inactive (no replicated data is captured, sent or received) on the replication servers in the specified *at_server_group* list. In addition, this command deletes any data in the send queue for the stopped replicate. You cannot stop replicates that have no participants.

If you omit the *at_server_group* list, the replicate enters the inactive state on all database servers participating in the replicate and all send queues for the replicate are deleted.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr stop replicate** to stop that individual replicate. You must use **cdr stop replicateset** to stop all replicates in the exclusive replicate set.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command with the **--repair** option.

When you run the **cdr stop replicate** command, an event alarm with a class ID of 61 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following command connects to the database server **lake** and stops the replicate **aRepl** on server groups **g_server1** and **g_server2**:

```
cdr sto rep -c lake aRepl g_server1 g_server2
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr suspend replicate on page 471

Enterprise Replication Event Alarms on page 221

Related information

Resynchronizing Data among Replication Servers on page 177

Enterprise Replication Server administrator on page 18

cdr stop replicateset

The **cdr stop replicateset** command stops capture and transmittal transactions for all the replicates in a replicate set.

Syntax

cdr stop replicateset [<Connect Option > (explicit id)] *repl_set* [*server_group*]

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to stop.	The replicate set must exist	Long Identifiers on page 256
server_group	Name of database server group on which to stop the replicate group.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicateset** command causes all replicates in the replicate set *repl_set* to enter the inactive state (no capture, no send) on the database servers in the *server_group* list.

If the *server_group* list is omitted, the replicate set *repl_set* enters the inactive state for all database servers participating in the replicate set.

If not all the replicates in the non-exclusive replicate set are active, the **cdr stop replicateset** command displays a warning and only stops the replicates that are currently active.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command.

When you run the **cdr stop replicateset** command, an event alarm with a class ID of 62 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the database server **paris** and stops the replicate set **accounts_set** on server groups **g_utah** and **g_iowa**:

```
cdr sto replset --connect=paris accounts_set g_utah g_iowa
```

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr define replicate on page 342

cdr suspend replicateset on page 472

Enterprise Replication Event Alarms on page 221

Related information

Enterprise Replication Server administrator on page 18

cdr suspend replicate

The **cdr suspend replicate** command suspends delivery of replication data.

Syntax

`cdr suspend replicate` [`<Connect Option >` (explicit id)] *repl_name*

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate.	The replicate must be active.	Long Identifiers on page 256

Usage

The **cdr suspend replicate** command causes the replicate *repl_name* to enter the suspend state (capture, no send) for all participants.

Attention: When a replicate is suspended, Enterprise Replication holds the replication data in the send queue until the replicate is resumed. If a large amount of data is generated for the replicate while it is suspended, the send queue space can fill, causing data to be lost. Enterprise Replication does not synchronize transactions if a replicate is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr suspend replicate** to suspend that individual replicate. You must use **cdr suspend replicateset** to suspend all replicates in the exclusive replicate set.

When you run the **cdr suspend replicate** command, an event alarm with a class ID of 55 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the database server **stan** and suspends the replicate **house**:

```
cdr sus repl --connect=stan house
```

Related reference

cdr change replicate on page 299

cdr define replicate on page 342

cdr delete replicate on page 374

cdr list replicate on page 395

cdr modify replicate on page 415

cdr resume replicate on page 440

cdr start replicate on page 445

cdr stop replicate on page 468

Enterprise Replication Event Alarms on page 221

cdr suspend replicateset on page 472

Related information

Enterprise Replication Server administrator on page 18

cdr suspend replicateset

The **cdr suspend replicateset** command suspends delivery of replication data for all the replicates in a replicate set.

Syntax

```
cdr suspend replicateset [ <Connect Option > (explicit id ) ] repl_set
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to suspend.	The replicate set must exist.	Long Identifiers on page 256

Usage

The **cdr suspend replicateset** command causes all the replicates in the replicate set *repl_set* to enter the suspend state. Information is captured, but no data is sent for any replicate in the set. The data is queued to be sent when the set is resumed.

Attention: When a replicate set is suspended, Enterprise Replication holds the replication data in the send queue until the set is resumed. If a large amount of data is generated for the replicates in the set while it is suspended, the send queue space can fill, causing data to be lost. Enterprise Replication does not synchronize transactions if a replicate in a replicate set is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.

If not all the replicates in the non-exclusive replicate set are active, the **cdr suspend replicateset** command displays a warning and only suspends the replicates that are currently active.

When you run the **cdr suspend replicateset** command, an event alarm with a class ID of 56 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default database server specified by **$INFORMIXSERVER** and suspends the replicate set **accounts_set**:

```
cdr sus replset account_set
```

Related reference

cdr change replicateset on page 302

cdr define replicateset on page 355

cdr delete replicateset on page 376

cdr list replicateset on page 400

cdr modify replicateset on page 419

cdr resume replicateset on page 441

cdr start replicateset on page 448

cdr stop replicateset on page 469

cdr define replicate on page 342

Enterprise Replication Event Alarms on page 221

cdr suspend replicate on page 471

Related information

Enterprise Replication Server administrator on page 18

cdr suspend server

The **cdr suspend server** command suspends the delivery of replication data to a database server from either a specified list of database servers or from all database servers in the domain.

Syntax

```
cdr suspend server [ <Connect Option > (explicit id ) ] to_server_group [ from_server_group ]
```

Element	Purpose	Restrictions
to_server_group	Name of database server group to which to suspend delivery of replication data.	The database server group must be currently active in Enterprise Replication.
from_server_group	Name of the database server group from which to stop sending data to *to_server_group*.	The database server group must be currently active in Enterprise Replication.

Usage

The **cdr suspend server** command suspends delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command suspends replication of data from all database servers participating in the replication domain to the *to_server_group*.

Suspending replication has the following effects:

- The connections between the suspended server and active replication servers remain active.
- Transactions on the suspended replication server are sent to the active replication servers.
- Transactions on active replication servers are queued for the suspended replication server.
- Control messages on active replication servers are sent to the suspended replication server.
- Control messages on the suspended replication server are sent to the active replication servers.

To restart replication on a suspended replication server, run the cdr resume server command. Shutting down and restarting the suspended database server does not resume replication.

When you run the **cdr suspend server** command, an event alarm with a class ID of 51 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) and suspends replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

```
cdr sus serv g_iowa g_ohio g_utah
```

Related reference

cdr connect server on page 337

cdr define server on page 357

cdr delete server on page 377

cdr disconnect server on page 385

cdr list server on page 402

cdr modify server on page 421

cdr resume server on page 442

Enterprise Replication Event Alarms on page 221

Related information

Enterprise Replication Server administrator on page 18

cdr swap shadow

The **cdr swap shadow** command switches a replicate with its shadow replicate during manual remastering.

Syntax

```
cdr swap shadow [ <Connect Option > (explicit id ) ] --primaryname=repl_name --primaryid=repl_ID --
shadowname=shadow_name --shadowid=shadow_ID
```

Element	Purpose	Restrictions	Syntax
repl_name	Name of the primary replicate.	The primary replicate participant attributes state, type (**P** or **R**), and table owner (**O** or **I**) must match the shadow replicate participant attributes.	Long Identifiers on page 256
repl_ID	Internal Enterprise Replication identification code for the primary replicate.		
shadow_name	Name of the shadow replicate.	The shadow replicate state must match the primary replicate state. Shadow replicate participants must match the primary replicate participants.	Long Identifiers on page 256
shadow_ID	Internal Enterprise Replication identification code for the shadow replicate.		

The following table describes the **cdr swap shadow** options.

Long Form	Short Form	Meaning
--primaryname=	**-p**	Specifies the name of the primary replicate.
--primaryid=	**-P**	Specifies the ID of the primary replicate.
--shadowname=	**-s**	Specifies the name of the shadow replicate.
--shadowid=	**-S**	Specifies the ID of the shadow replicate.

Usage

Use the **cdr swap shadow** command to switch a replicate with its shadow replicate as the last step in manually remastering a replicate that was created with the **--name=n** option. You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option.

Use the **onstat -g cat repls** command to obtain the *repl_ID* and *shadow_ID*. Alternatively, you can query the **syscdrrepl** view in the **sysmaster** database.

You can run this command from within an SQL statement by using the SQL administration API.

Related reference

cdr alter on page 290

cdr define replicate on page 342

cdr list replicate on page 395

Participant and participant modifier on page 257

onstat -g rep: Prints the schedule manager queue on page 566

Related information

Remastering replicates without name verification on page 195

Enabling code set conversion between replicates on page 108

Enterprise Replication Server administrator on page 18

cdr sync replicate

The cdr sync replicate command synchronizes data among replication servers to repair inconsistent data within a replicate.

Syntax

```
cdr sync replicate [ <Connect Option > (explicit id ) ] --master = data_server --repl=repl_name { target_server | --all } [ --name=task_name ] [ --extratargetrows= { delete | keep | merge } ] [ --firetrigger= { off | on | follow } ] [ --memadjust =size { K | M } ] [ --background ] [ --excludeTimeSeries ] [ --ignoreHiddenTSElements ]
```

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
repl_name	Name of the replicate to synchronize.		Long Identifiers on page 256
*size*K or *size*M	Size, in either kilobytes (K) or megabytes (M), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
target_server	Name of a database server group on which to perform synchronization.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers on page 256

The following table describes the cdr sync replicate options.

Long Form	Short Form	Meaning
--all	-a	Specifies that all servers defined for the replicate are checked.
--background	-B	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the --name= option, or the time stamp for the command if --name= is not specified.
--excludeTimeSeries		Specifies to prevent the checking of time series data.
--extratargetrows=	-e	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*):

Long Form	Short Form	Meaning

		• delete: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers
		• keep: retain rows on the target servers
		• merge: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns.
		Note: When **cdr sync replicate** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation:
		`WARNING: Extra row option isn't applicable to`
		` Send-Only participant.`
		`'--extratargetrows' option is ignored`
		Note: When **cdr syn replicate** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted:
		`Error: Receive only participant '%s' can not be`
		` a master`
		`node in data synchronization task`

--firetrigger=	-T	Specifies how to handle triggers at the target servers while data is synchronizing: • off: (default) do not fire triggers at target servers during synchronization • on: always fire triggers at the target servers even if the replicate definition does not have the --firetrigger option • follow: fire triggers at target servers only if the replicate definition has the --firetrigger option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--master=	-m	Specifies the database server to use as the reference copy of the data.

Long Form	Short Form	Meaning
--memadjust=	-J	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--name=	-n	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--repl=	-r	Specifies the name of the replicate to synchronize.

Usage

Use the cdr sync replicate command to synchronize data between multiple database servers for a specific replicate. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the --memadjust option to specify the size of the send queue.

If you want to monitor the progress of the synchronization operation, include the --name option and specify a name for the progress report task. Then run the cdr stats sync command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the --name option so that you can monitor the operation with the cdr stats sync command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

The cdr sync replicate command performs the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs a sequential scan of the replicated table on the source server.
3. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs. Rows are not replicated to participants that include the S option in the participant definition because those participants only send data.
4. Deletes the shadow replicate.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information on these error codes, see Return Codes for the cdr Utility on page 262

Example

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate named **repl_1**:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--all --extratargetrows=keep\
--firetrigger=on
```

The data on the server group **g_serv1** is used as the reference for correcting the data on the other servers. Line 2 indicates that all servers associated with the replicate are synchronized and that if the synchronization operation detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers. Line 3 indicates that triggers should be fired on the target servers even if the replicate definition does not include the --firetrigger option.

Example

Example 2: Synchronize three servers

The following example illustrates synchronizing three servers for the replicate named **repl_2**:

```
cdr sync replicate -m g_serv1 -r repl_2\
g_serv2 g_serv3
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the --extratargetrows option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--memadjust=50M --background
```

Related reference

cdr check replicate on page 313

cdr stats sync on page 461

Related information

Performing Direct Synchronization on page 178

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr sync replicateset

ASDF The cdr sync replicateset command synchronizes data among replication servers to repair inconsistent data within a replicate set.

Syntax

```
cdr sync replicateset [ <Connect Option > (explicit id ) ] --master = data_server { --replset = repl_set | --allrepl } {
target_server | --all } [ --name=task_name ] [ --extratargetrows= { delete | keep | merge } ] [ --firetrigger= { off |
on | follow } ] [ --memadjust = size { K | M } ] [ --background ] [ --process= number_processes ] [ --excludeTimeSeries ] [
 --ignoreHiddenTSElements ]
```

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256
number_processes	The number of parallel processes to use for the command.	The maximum number of processes Enterprise Replication can use is equal to the number of replicates in the replicate set.	
repl_set	Name of the replicate set. Can be the name of a derived replicate set.		Long Identifiers on page 256
seconds	The number of seconds to wait for a disabled replication server to be recognized as active by other replication servers in the domain and how long to wait for control messages queued at peer servers to be applied at newly-enabled server.	Must be an integer value from 0 to 60.	
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers on page 256

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers on page 256

The following table describes the cdr sync replicateset options.

Long Form	Short Form	Meaning
--all	-a	Specifies that all servers defined for the replicate are checked.
--allrepl	-A	Specifies that all replicates are synchronized.
--excludeTimeSeries		Specifies to prevent the checking of time series data.

--extratargetrows=	-e	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*):
		• delete: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers
		• keep: retain rows on the target servers
		• merge: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns.
		Note: When **cdr sync replicateset** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation:
		`WARNING: Extra row option isn't applicable`
		` to Send-Only participant.`
		`'--extratargetrows' option is ignored`
		Note: When **cdr syn replicateset** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted:

Long Form	Short Form	Meaning

		✎ `Error: Receive only participant '%s’ can not`
		` be a master`
		`node in data synchronization task`

--firetrigger=	-T	Specifies how to handle triggers at the target servers while data is synchronizing: • off: (default) do not fire triggers at target servers during synchronization • on: always fire triggers at the target servers even if the replicate definition does not have the --firetrigger option • follow: fire triggers at target servers only if the replicate definition has the --firetrigger option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--master=	-m	Specifies the database server to use as the reference copy of the data.
--memadjust=	-J	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--name=	-n	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--process=	-p	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--replset	-s	Specifies the name of the replicate set to synchronize.

Usage

Use the cdr sync replicateset command to synchronize data between multiple database servers for a replicate set. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the --memadjust option to specify the size of the send queue.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the --process option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If you want to monitor the progress of the synchronization operation, include the --name option and specify a name for the progress report task. Then run the cdr stats sync command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the --background option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the --name option so that you can monitor the operation with the cdr stats sync command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

To synchronize all replicates at once, use the --allrepl option.

The cdr sync replicateset command performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
3. Performs a sequential scan of the replicated table on the source server.
4. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs. Rows are not replicated to participants that include the S option in the participant definition because those participants only send data.
5. Deletes the shadow replicate.
6. Repeats steps 2 through 5 for each replicate in the replicate set.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information on these error codes, see Return Codes for the cdr Utility on page 262

Example

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate set **replset_1** using **g_serv1** as the reference server:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--all --extratargetrows=keep
```

Line 2 indicates that all servers associated with the replicate set are synchronized and that if the synchronization process detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers.

Example

Example 2: Synchronize three servers in parallel

The following example illustrates synchronizing three servers for the replicate set named **replset_2** and using two processes to synchronize each of the two replicates in the set in parallel:

```
cdr sync replicateset -m g_serv1 -s replset_2\
g_serv2 g_serv3 --process=2
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the --extratargetrows option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--memadjust=50M --background
```

Example

Example 4: Synchronize all replicate sets on a replication server

The following command synchronizes all replicate sets on a replication server named **g_serv2**:

```
cdr sync replicateset --allrepl g_serv2
```

The replicate set name is not specified because the --allrepl option is used.

Related reference

cdr check replicateset on page 325

cdr stats sync on page 461

Related information

Altering multiple tables in a replicate set on page 190

Performing Direct Synchronization on page 178

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

cdr -V

The **cdr -V** command displays the version of Informix® that is currently running.

Syntax

```
cdr -V
```

Usage

Use the **cdr -V** command if you need to obtain the version of the database server, usually at the request of IBM® Software Support.

Example

Examples

The following example shows an example output of the **cdr -V** command:

```
IBM Informix Version 11.70.UC1    Software Serial Number RDS#N000000
```

Related information

Enterprise Replication Server administrator on page 18

cdr view

The cdr view command shows information about every Enterprise Replication server in the domain.

Syntax

```
cdr view [ <Connect Option > (explicit id ) ] { { { | state | profile | ddr [ --logstage] | servers | sendq | rcv | apply |
nif | ats | ris <ATS and RIS Directory Options> } [ --repeat=time ] } | [ --help] }
```

ATS and RIS Directory Options

```
“{ | --atsdir | --risdir }”

“[{ --repair { --verbose | [ --quiet] } [ --delete] | --check }]”
```

Element	Purpose	Restrictions
time	The number of seconds before the cdr view command is repeated.	Must be a positive integer.

The following table describes the cdr view subcommands.

Long Form	Meaning
apply	Show a summary of how data is being applied on each of the target servers, including the latency of each target server.
ats	Show a portion of each ATS file that is in text format.
atsdir	Show the names of the files in the ATS directory that are in text format and optionally run repair operations that are based on those files. If you are running this command as a DBSA, you must have read permission on the ATS files. Permissions on ATS files can be set with the chown operating system command.
ddr	Show the state, key log positions, and the proximity to transaction blocking for each server in the replication domain.
nif	Show information about the network connections between Enterprise Replication servers, including the number of transactions that are waiting to be transmitted to target servers.
profile	Show a summary of the state, data capture, data apply, errors, connectivity, queues, and the size of spooling files for every Enterprise Replication server.
rcv	Show information about the receive statistics for each target server, including the number of transaction failures and the rate at which transactions are applied.
ris	Show a portion of each RIS file that is in text format.
risdir	Show the names of the files in the RIS directory that are in text format and optionally run repair operations that are based on those files.

Long Form	Meaning
	If you are running this command as a DBSA, you must have read permission on the RIS files. Permissions on RIS files can be set with the chown operating system command.
sendq	Show information about the send queues for each Enterprise Replication server.
servers	Show information about the state, connection status to each peer server, and queue size for each Enterprise Replication server.
state	Show the Enterprise Replication state and the state of data capture, network connections, and data apply for each Enterprise Replication server.

The following table describes the cdr view options.

Long Form	Short Form	Meaning
--check	-C	Check the consistency between the database server and the ATS or RIS file. Send repair operations to stderr, but do not perform the repair operations.
--delete	-d	Delete ATS or RIS files after processing them with the repair operation.
--help	-h	Show the cdr view command usage.
-logstage	-l	Show log staging statistics.
--quiet	-q	Quiet mode. Repair operations are not sent to stderr.
--repair	-R	Synchronize data based on ATS or RIS files in text format.
--repeat=	-r	Repeat the cdr view command after the number of seconds specified by the *time* element.
--verbose	-v	Verbose mode (default). All repair operations are sent to stderr.

Usage

Use the cdr view command to monitor the Enterprise Replication domain. Each subcommand results in different output information.

You can choose to show the output of multiple subcommands sequentially by including them in the same cdr view command. You can choose to automatically repeat the command by using the --repeat option to specify the seconds in between commands.

You can repair inconsistencies that are listed in ATS or RIS files on every server by using the --repair option. Use the --delete option to delete the ATS or RIS files after the repair is complete.

Tip: Using the --repair option is equivalent to running the cdr repair command. The --check option is equivalent to the cdr repair --check command.

The cdr view state Command Output

The following example of the output of the cdr view state command shows the state of Enterprise Replication and each of its main components for every server in the Enterprise Replication domain.

```
STATE
Source      ER                Capture           Network           Apply
            State             State             State             State
-------------------------------------------------------------------------------
cdr1        Active            Running           Running           Running
cdr2        Active            Running           Running           Running
cdr3        Active            Running           Running           Running
cdr4        Active            Running           Running           Running
```

In this example, Enterprise Replication is active and running normally on all servers.

Possible values in the ER State column include:

Abort

Enterprise Replication is aborting on this server.

Active

Enterprise Replication is running normally.

Down

Enterprise Replication is stopped.

Dropped

The attempt to drop the **syscdr** database failed.

Init Failed

The initial start of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server.

Initializing

Enterprise Replication is being defined.

Initial Startup

Enterprise Replication is starting for the first time on this server.

Shutting Down

Enterprise Replication is stopping on this server.

Startup Blocked

Enterprise Replication cannot start because the server was started with the oninit -D command.

Synchronizing Catalogs

The server is receiving a copy of the **syscdr** database.

Uninitialized

The server does not have Enterprise Replication defined on it.

Possible values in the Capture State, Network State, and Apply State columns include:

Running

The Enterprise Replication component is running normally.

Down

The Enterprise Replication component is not running.

Uninitialized

The server is not a source server for replication.

The cdr view profile Command Output

The following example of the output of the cdr view profile command shows a summary of the other cdr view commands and information about the sbspaces that are designated for spooled transaction data.

```
ER PROFILE for Node cdr2                ER State Active

DDR - Running                           SPOOL DISK USAGE
  Current        4:16879616               Total                   100000
  Snoopy         4:16877344               Metadata Free             5025
  Replay         4:24                     Userdata Free            93193
  Pages from Log Lag State   43879
                                        RECVQ
SENDQ                                     Txn In Queue                 0
  Txn In Queue               0            Txn In Pending List          0
  Txn Spooled                0
  Acks Pending               0          APPLY - Running
                                          Txn Processed             1838
NETWORK - Running                         Commit Rate              76.58
  Currently connected to 3 out of 3       Avg. Active Apply         1.16
  Msg Sent                1841            Fail Rate                 0.00
  Msg Received            5710            Total Failures               0
  Throughput           1436.94            Avg Latency               0.00
  Pending Messages           0            Max Latency                  0
                                          ATS File Count               0
                                          RIS File Count               0
```

In this example, only the output for a single server, **cdr2**, is shown. The actual output of the cdr view profile command includes a similar profile for every server.

The DDR section is a summary of the cdr view ddr command.

The SPOOL DISK USAGE section shows the total amount of memory, in bytes, in the sbspaces that Enterprise Replication uses to store spooled transaction row data, and the amount of available metadata and user data space.

The SENDQ section is a summary of the cdr view sendq command.

The RECVQ section is a summary of the cdr view rcv command.

The NETWORK section is a summary of the cdr view nif command.

The APPLY section is a summary of the cdr view apply command.

The cdr view ddr Command Output

The following example of the output of the cdr view ddr command shows the status of log capture.

```
Server   Snoopy       Replay       Current      total        log pages to   LogLag   Cur LogLag
         log page     log page     log page     log pages    LogLag State   State    Action
-------------------------------------------------------------------------------------------
g_bombay 16:133       16:0         16:134       30000        17866          Off       dlog
g_delhi  30:490       30:0         30:491       5000         3508           Off       logstage
```

The following example of the output of the cdr view ddr -l command shows the status of log capture.

```
Server   Disk Space   Max allowed  Max disk     Cur Staged
         Usage(%)     Space(KB)    ever used(KB) log file cnt
-------------------------------------------------------------------------------------------
g_bombay 0.00         0            0.00         0
g_delhi  0.00         1048576      0.00         0
```

The columns in the output of the cdr view ddr command provide the following information:

Server

The name of the Enterprise Replication server.

Snoopy log page

The current log ID and position at which transactions are being captured for replication.

Replay log page

The current log ID and position at which transactions have been applied. This is the position from which the log would must be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.

Current® log page

The log page on which replicated transactions are being captured.

total log pages

The total number of log pages on the server.

log pages to LogLag State

The number of log pages that must be used before transaction blocking occurs.

LogLag State

The state of DDR log lag: on or off.

Cur LogLag Action

The action being taken to catch up logs.

For more information on interpreting this output, see onstat -g ddr: Print status of ER log reader on page 553.

The cdr view servers Command Output

The following example of the output of the cdr view servers command shows the state of the Enterprise Replication servers and their connections to each other.

```
SERVERS
Server Peer   ID   State    Status     Queue  Connection Changed
---------------------------------------------------------------------------
cdr1   cdr1   1    Active   Local      0
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Connected  0      Apr 15 10:46:15
cdr2   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Local      0
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Connected  0      Apr 15 10:46:16
cdr3   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Local      0
       cdr4   4    Active   Connected  0      Apr 15 10:46:16
cdr4   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Local      0
```

In this example, each of the four servers is connected to each other.

The output of this command is similar to the output of the cdr list server command, except that the cdr view server command shows all servers in the Enterprise Replication domain, not just the servers connected to the one from which the command is run. For information about the columns in this output, see cdr list server on page 402.

The cdr view sendq Command Output

The following example of the output of the cdr view sendq command shows information about the send queue for each server.

```
RQM SENDQ
Server     Trans.     Trans.     Trans.      Data     Memory      ACKS
           in que     in mem    spooled  in queue     in use   pending
---------------------------------------------------------------------------
cdr1          594        594          0     49896      49896         0
cdr2            0          0          0         0          0         0
cdr3            0          0          0         0          0         0
cdr4            0          0          0         0          0         0
```

In this example, only the server **cdr1** has transactions in the send queue, all of which are in memory.

The columns of the cdr view sendq command provide the following information in addition to the server name:

Trans. in que

The number of transactions in the send queue.

Trans. in mem

The number of transactions in the send queue that are currently in memory.

Trans. spooled

The number of transactions in the send queue that have been spooled to disk.

Data in queue

The number of bytes of data in the send queue, including both in-memory and spooled transactions.

Memory in use

The number of bytes of data in the send queue that resides in memory.

ACKS pending

The number of acknowledgments that have been received but have not yet been processed.

The cdr view rcv Command Output

The following example of the output of the cdr view rcv command shows information about the receive queue for each server.

```
RCV
Server Received Spooled    Memory Pending Waiting
           Txn.    Txn.    In Use    Txn.    Txn.
---------------------------------------------------------------------------
cdr1          0       0         0       0       0
cdr2        372       0    871164     372       0
cdr3        220       0     18480     220       0
cdr4          0       0         0       0       0
```

In this example, the servers **cdr2** and **cdr3** have transactions in the receive queue, all of which have been preprocessed and are in the pending state waiting to be applied.

The columns of the cdr view rcv command provide the following information in addition to the server name:

Received Txn.

The number of transactions in the receive queue.

Spooled Txn.

The number of transactions in the receive queue that have been spooled to disk.

Memory In Use

The size, in bytes, of the receive queue.

Pending Txn.

The number of transactions that have been preprocessed but not yet applied.

Waiting Txn.

The number of acknowledgments waiting to be sent back to the source server.

The cdr view apply Command Output

The following example of the output of the cdr view apply command shows how replicated data is being applied.

```
APPLY
Server   Pl  Failure       Num     Num   Apply  --Latency--  ATS  RIS
       Rate    Ratio       Run  Failed    Rate  Max    Avg.    #    #
---------------------------------------------------------------------------
cdr1      0    0.000         0       0   0.000    0   0.000    0    0
cdr2      0    0.000     10001       0   0.112    0   0.000    0    0
cdr3      0    0.000     10001       0   0.112    0   0.000    0    0
cdr4      0    0.000     10001       0   0.112    0   0.000    0    0
```

In this example, the servers **cdr2**, **cdr3**, and **cdr4** each applied 10 001 transactions.

The columns of the cdr view apply command provide the following information in addition to the server name:

Pl Rate

Indicates the degree of parallelism used when data is being applied. Zero indicates the highest possible rate of parallelism.

Failure Ratio

The ratio of the number of times data could not be applied in parallel because of deadlocks or lock time outs.

Num Run

The number of transactions processed.

Num Failed

The number of failed transactions because of deadlocks or lock time outs.

Apply Rate

The number of transactions that have been applied divided by the amount of time that replication has been active. The Apply Rate is equal to the Commit Rate in the cdr view profile command.

Max. Latency

The maximum number of seconds for processing any transaction.

Avg. Latency

The average number of seconds of the lifecycle of a replicated transaction.

ATS #

The number of ATS files.

RIS #

The number of RIS files.

The cdr view nif Command Output

The following example of the output of the cdr view nif command shows the status and statistics of connections between servers.

```
NIF
Source Peer   State         Messages  Messages Messages  Transmit
                                Sent  Received  Pending      Rate
---------------------------------------------------------------------------
cdr1   cdr2   Connected        24014       372        6 21371.648
       cdr3   Connected        24020        17        0 20527.105
       cdr4   Connected        24014        23        6 21925.727
cdr2   cdr1   Connected          392     24015        0 21380.879
       cdr3   Connected           14        14        0    10.857
       cdr4   Connected           14        14        0    11.227
cdr3   cdr1   Connected           17     24021        0 20310.611
       cdr2   Connected           14        14        0    10.739
       cdr4   Connected           14        14        0    11.227
cdr4   cdr1   Connected          236     24015        0 21784.225
       cdr2   Connected           14        14        0    11.101
       cdr3   Connected           14        14        0    11.101
```

In this example, all servers are connected to each other. The server **cdr1** has six messages that have not yet been sent to server **cdr2** and server **cdr4**.

The columns of the cdr view nif command provide the following information in addition to the source server name:

Peer

The name of the server to which the source server is connected.

State

The connection state. Values include:

Connected

The connection is active.

Disconnected

The connection was explicitly disconnected.

Timeout

The connection attempt has timed out, but will be reattempted.

Logic error

The connection disconnected due to an error during message transmission.

Start error

The connection disconnected due to an error while starting a thread to receive remote messages.

Admin close

Enterprise Replication was stopped by a user issuing the cdr stop command.

Connecting

The connection is being established.

Never Connected

The servers have never had an active connection.

Messages Sent

The number of messages sent from the source server to the target server.

Messages Received

The number of messages received by the source server from the target server.

Messages Pending

The number of messages that the source server must send to the target server.

Transmit Rate

The total bytes of messages sent and received by the server divided by the amount of time that Enterprise Replication has been running. Same as the Throughput field in the cdr view profile command.

The cdr view ats and cdr view ris Command Output

The following example of the output of the cdr view ats command shows that there are no ATS files in text format.

```
ATS for cdr1 - no files
----------------------------------------------------------------------------

ATS for cdr2 - no files
----------------------------------------------------------------------------

ATS for cdr3 - no files
----------------------------------------------------------------------------

ATS for cdr4 - no files
```

The following example of the cdr view ris command shows two RIS files in text format.

```
RIS for cdr1 - no files
----------------------------------------------------------------------------

RIS for cdr2 - 1 files
Source Txn. Commit         Receive
       Time                Time
----------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr2.cdr1.D_4.080415_11:56:14.1

Row:2 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
----------------------------------------------------------------------------

RIS for cdr3 - no files
----------------------------------------------------------------------------
```

```
RIS for cdr4 - 1 files
Source Txn. Commit         Receive
       Time                Time
---------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr4.cdr1.D_1.080415_11:56:14.1

Row:3 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
```

In this example, the servers **cdr2** and **cdr4** each have one RIS file.

The cdr view atsdir and cdr view risdir Command Output

The cdr view atsdir command and cdr view risdir command outputs have the same format. The following example of the output of the cdr view risdir command shows the names of two RIS files.

```
RISDIR
Server File                                  Size               Create
       Name                                                       Time
---------------------------------------------------------------------------
cdr2   ris.cdr2.cdr1.D_4.080415_11:56:14.1    465  2008-04-15 11:56:15
cdr4   ris.cdr4.cdr1.D_1.080415_11:56:14.1    475  2008-04-15 11:56:15
```

In this example, both server **cdr2** and server **cdr4** have a single RIS file. The Size column shows the size of the file, in bytes.

Example

Examples

The following command would show information about the send queue and the network every 10 seconds:

```
cdr view sendq nif --repeat=10
```

The following command can be used in a daemon or script that runs every five minutes to check all servers for ATS and RIS files, repair inconsistencies, and delete the processed ATS and RIS files:

```
cdr view atsdir risdir --repair --delete --repeat=300
```

Related reference

cdr list server on page 402

cdr repair on page 435

onstat -g ddr: Print status of ER log reader on page 553

Related information

Failed Transaction (ATS and RIS) Files on page 199

Monitor and troubleshooting Enterprise Replication on page 197

Preparing for Role Separation (UNIX) on page 80

Enterprise Replication Server administrator on page 18

Enterprise Replication configuration parameter and environment variable reference

You can use configuration parameters and environment variables to configure the behavior of Enterprise Replication.

The database server onconfig configuration file includes the configuration parameters that affect the behavior of Enterprise Replication. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, the DBSERVERNAME configuration parameter must specify the network connection and not to a shared-memory connection. For information about database server aliases, see the *HCL® Informix® Administrator's Guide*.

Use the CDR_ENV configuration parameter to set the environment variables that affect the behavior of Enterprise Replication.

You can view the setting of Enterprise Replication configuration parameters and environment variables with the onstat -g cdr config command. See onstat -g cdr config: Print ER settings on page 551.

Related reference

Set configuration parameters for replication on page 72

Related information

Dynamically Modifying Configuration Parameters for a Replication Server on page 162

Configuring network encryption for replication servers on page 61

CDR_APPLY Configuration Parameter

Specifies the minimum and maximum number of data sync threads. The value is updated dynamically as needed.

onconfig.std value

Not in the `onconfig.std` file.

range of values

Two positive integers that are separated by a comma and that represent the minimum and maximum number of threads per CPU virtual processor. If only one value is specified, that value is the maximum and the minimum is set to 1.

If you do not set the CDR_APPLY configuration parameter explicitly, Enterprise Replication automatically allocates data sync threads for each CPU VP based on need:

- The minimum number of data sync threads is the number of available CPU VPs.
- The maximum number of data sync threads is four times the number of available CPU VPs.

You can set the CDR_APPLY configuration parameter to control the minimum and maximum configurable data sync apply threads irrespective of the CPU VP configuration.

CDR_AUTO_DISCOVER configuration parameter

Use the CDR_AUTO_DISCOVER configuration parameter to enable connectivity autoconfiguration for a high-availability cluster or Enterprise Replication domain, or to autoconfigure replication.

onconfig.std value

CDR_AUTO_DISCOVER 0

default value if you created a server during installation

CDR_AUTO_DISCOVER 1

values

0 = Disable connectivity and Enterprise Replication autoconfiguration.

1 = Enable connectivity and Enterprise Replication autoconfiguration.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

After you run the SQL administration API task() or admin() function with the -wf CDR_AUTO_DISCOVER=*value* or -wm CDR_AUTO_DISCOVER=*value* arguments.

Usage

When the CDR_AUTO_DISCOVER configuration parameter is set to 1, the following commands are enabled:

- cdr autoconfig serv, which autoconfigures connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can autoconfigure replication.
- ifxclone with the --autoconf option, which autoconfigures connectivity information between a newly added server and the other servers of a high-availability cluster or Enterprise Replication domain. If you use the ifxclone utility to create an Enterprise Replication server, the --autoconf option can autoconfigure replication.

Related reference

cdr autoconfig serv on page 291

Related information

cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API) on page

The ifxclone utility on page

Preparing the Network Environment on page 55

CDR_DBSPACE Configuration Parameter

Specifies the dbspace where the **syscdr** database is created.

onconfig.std value

none

units

any valid dbspace

takes effect

When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DBSPACE configuration parameter specifies the dbspace where the **syscdr** database is created.

If the CDR_DBSPACE configuration parameter is not set, then **syscdr** is created in the root dbspace.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_DBSPACE configuration parameter when defining a replication server. If the CDR_DBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the cdr define command performs the following tasks:

- Creates a new dbspace using one or more new chunks from the storage pool
- Sets the CDR_DBSPACE configuration parameter both in memory and in the `onconfig` file to the newly defined dbspace.

For clusters, the cdr define command creates new dbspaces and sets the CDR_DBSPACE configuration parameters in all secondary database servers, as well.

Note: A database server's storage pool must have 200 MB of free space for the dbspace, and chunk sizes of 100 MB or greater for the database server to use automatic storage provisioning.

From 14.10xC6 onwards, row data for smaller transactions of size less than 26KB are stored in-line along with transaction header table trg_send_stxn. Row data for transaction size above 26KB is still stored in ER queue smart blob space configured using CDR_QDATA_SBSPACE configuration parameter. Make sure to have enough space allocated for storage space used to store trg_send_stxn table. trg_send_stxn table is created in space configured using CDR_DBSPACE configuration parameter. If CDR_DBSPACE is not configured, then rootdbs will be used for syscdr database and transaction header table.

CDR_DELAY_PURGE_DTC configuration parameter

Specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.

onconfig.std value

0

default value if not present in the onconfig

0

syntax

CDR_DELAY_PURGE_DTC *timeunit*

range of values

The range of values for *time* are 0 and positive integers.

The range of values for *unit* are:

- S = seconds (Default)
- M = minutes
- H = hours
- D = days

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

By default, rows in delete tables are deleted when those rows are no longer required by the timestamp conflict resolution rule. If you want to perform time stamp repair and your replicates use the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. The longer you retain rows in delete tables, the more accurate time stamp repairs are, but the more disk space the delete tables consume.

Tip: Right before you enable a disabled server, dynamically update the CDR_DELAY_PURGE_DTC configuration parameter to set it to a value slightly greater than the time that the server was disabled plus the amount of time a repair takes.

Related information

Repair inconsistencies by time stamp on page 184

Delete wins conflict resolution rule on page 48

onmode -wf, -wm: Dynamically change certain configuration parameters on page

CDR_DSLOCKWAIT Configuration Parameter

Specifies the number of seconds the data sync component waits for the database locks to be released.

onconfig.std value

`5`

units

seconds

takes effect

When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DSLOCKWAIT configuration parameter specifies the number of seconds the data sync component waits for database locks to be released. The CDR_DSLOCKWAIT parameter behaves similarly to the SET LOCK MODE statement. Although the SET LOCK MODE is set by the end user application, CDR_DSLOCKWAIT is used by Enterprise Replication while applying data at the target database. This parameter is useful in conditions where different sources require locks on the replicated table. These sources could be a replicated transaction from another server or a local application operating on that table.

Transactions that receive updates and deletes from another server in the replicate can abort because of locking problems. If you experience transaction aborts in the data sync due to lock timeouts like this, you might want to increase the value of this parameter.

CDR_ENV Configuration Parameter

Sets the Enterprise Replication environment variables CDR_ALARMS, CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, or CDR_RMSCALEFACT.

Important: Use the CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, and CDR_RMSCALEFACT environment variables only if instructed to do so by IBM® Support.

units

Enterprise Replication environment variable name and value, separated by an equal sign

takes effect

When the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command

The `onconfig` file can contain multiple entries for the CDR_ENV environment variable. You can specify only one environment variable per CDR_ENV entry.

Example

The following line in the `onconfig` file sets the CDR_ALARMS environment variable to add event alarm 51 to the event alarms that are enabled by default:

```
CDR_ENV CDR_ALARMS=30-39,47,48,50,51,71,73-75
```

When you update the CDR_ALARMS environment variable in the `onconfig` file, you must list all the Enterprise Replication event alarms that you want to be enabled.

The following lines in the `onconfig` file set the CDR_LOGDELTA environment variable to `30` and the CDR_ROUTER environment variable to `1`:

```
CDR_ENV CDR_LOGDELTA=30
CDR_ENV CDR_ROUTER=1
```

Related information

Enabling or Disabling Enterprise Replication Event Alarms on page 244

CDR_EVALTHREADS Configuration Parameter

Specifies the number of group evaluator threads to create when Enterprise Replication starts, and enables parallelism.

onconfig.std value

`1,2`

units

evaluator thread instances

range of values

first value: `0` or a positive integer representing the number of evaluator threads to create per CPU VP. Although evaluator threads are not assigned to specific CPU VPs, you can create evaluator threads that are proportional in number to the number of CPU VPs.

second value: `0` or a positive integer representing the additional number of evaluator threads to create irrespective of the number of CPU VPs.

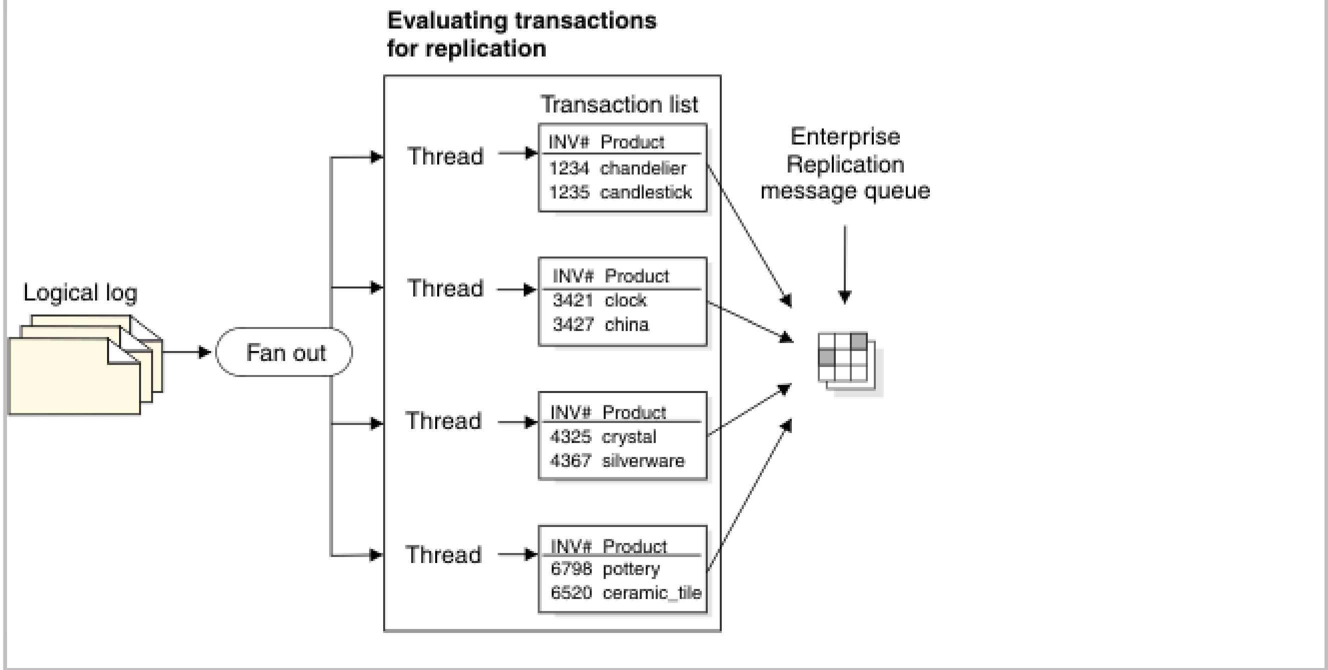
Do not set both CDR_EVALTHREADS values to 0.

takes effect

When the database server is shut down and restarted, or immediately after the cdr change onconfig command is used

Enterprise Replication evaluates the images of a row in parallel to assure high performance. Figure 25: Processing in Parallel for High Performance on page 504 illustrates how Enterprise Replication uses parallel processing to evaluate transactions for replication.

Figure 25. Processing in Parallel for High Performance



The CDR_EVALTHREADS configuration parameter specifies the number of grouper evaluator threads to create when Enterprise Replication starts and enables parallelism. The format is:

```
(per-cpu-vp,additional)
```

The following table provides four examples of CDR_EVALTHREADS.

Number of Threads	Explanation	Example
`1,2`	1 evaluator thread per CPU VP, plus 2	For a 3 CPU VP server: (3 * 1) + 2 = 5
`2`	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3 * 2) = 6
`2,0`	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3* 2) +0 = 6
`0,4`	4 evaluator threads for any database server	For a 3 CPU VP server: (3 * 0) +4 = 4

Attention: Do not configure the total number of evaluator threads to be smaller than the number of CPU VPs in the system. As noted above, do not set both CDR_EVALTHREADS values to `0`.

CDR_LOG_LAG_ACTION configuration parameter

Specifies how Enterprise Replication responds to a potential log wrap situation.

onconfig.std value

CDR_LOG_LAG_ACTION ddrblock

separators

+

range of values

See the Usage section.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

Usage

Use the CDR_LOG_LAG_ACTION configuration parameter to specify one or more actions, in priority order, that Enterprise Replication takes during a potential log wrap situation.

Syntax for the CDR_LOG_LAG_ACTION configuration parameter

```
CDR_LOG_LAG_ACTION { { logstage [ +dlog ] | [ dlog [ +logstage ] ] } [ { +ignore | +ddrblock | +shutdown } ] | { ignore | ddrblock |
shutdown } }
```

Table 34. Options for the CDR_LOG_LAG_ACTION configuration parameter value

Option	Description
logstage	Enables compressed logical log staging. The following configuration parameters must also be set: • The LOG_STAGING_DIR configuration parameter must be set to a directory. The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission. • The CDR_LOG_STAGING_MAXSIZE configuration parameter must be set to a positive number. Log files are staged in the directory specified by the LOG_STAGING_DIR configuration parameter, until the maximum size specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is reached. The staged log files are deleted after advancing the log replay position. If the amount of disk space specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is exceeded, event alarm 30005 is raised. If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.

Table 34. Options for the CDR_LOG_LAG_ACTION configuration parameter value (continued)

Option	Description
dlog	Enables the dynamic addition of logical logs. The following configuration parameters must be set: • The CDR_MAX_DYNAMIC_LOGS configuration parameter must be set to -1 or a positive number. • The DYNAMIC_LOGS configuration parameter must be set to 2.
ignore	Ignore the potential for log wrapping. The Enterprise Replication replay position might be overrun. If the replay position is overrun, event alarm 30 is raised. Restart Enterprise Replication using the cdr cleanstart command and synchronize the data. The ignore option must be the only or the last option. If the snoopy log position overrun is detected, Enterprise Replication shuts down with event alarm 47005.
ddrblock	Default. Block client applications update activity. The ddrblock option must be the only or the last option.
shutdown	Shut down Enterprise Replication on the affected server. If replay position overrun is detected, restart Enterprise Replication using the cdr cleanstart command and synchronize the data. If the replay position was not overrun, restart Enterprise Replication using the cdr start command; there is no need to synchronize the data. If replay position overrun is detected and the cdr start command fails with error code 214 and raises event alarm class 75, restart Enterprise Replication using the cdr cleanstart command and synchronize the data. The shutdown option must be the only or the last option. If a log lag state is detected, Enterprise Replication is shut down and event alarm ID 47006 is raised.

Staged log file format

Enterprise Replication creates a directory named: `ifmxddrlog_`*`SERVERNUM`* in the directory specified by the LOG_STAGING_DIR configuration parameter. Log file names are in the following format:

```
ifmxERDDRBLOCKUniqueLog_lf_used_loguniqueid.dat
```

Enterprise Replication also creates an empty token file for each staged log file. The token file is used to detect log files that are only partially written. If a token file is not found then Enterprise Replication treats the staged log file as partially written log file and deletes it. The token log file format is:

```
ifmxERDDRBLOCKUniqueLog_lf_used_loguniqueid
```

Transferring log files to a high-availability cluster secondary server when using ER

If your configuration consists of an HDR, RSS, or SDS secondary server configured as an Enterprise Replication node, transfer staged log files to the secondary server using the alarm program script. The staged log files are required by Enterprise Replication in case the primary server in a high-availability cluster fails and a secondary server takes over the role of the primary server.

Enterprise Replication raises alarm class ID 30 and unique ID 30006 when a log is staged to the log staging directory. Enterprise Replication raises alarm class ID 30 and unique ID 30007 after deleting a staged log file. Using these alarms, you can automate the transfer of staged log files to the high-availability cluster secondary server using the alarm program script.

Ensure that the directory under the directory specified the LOG_STAGING_DIR configuration parameter exists and is named using the format `ifmxddrlog_`*`SERVERNUM`*. The script copies the staged log files to `ifmxddrlog_`*`SERVERNUM`* and creates a token log file after copying the staged log file.

Example

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by first staging compressed logs until they reach 1 MB in size, then dynamically add up to two logical logs, and then block user transactions. Set the following configuration parameters:

```
CDR_LOG_LAG_ACTION      logstage+dlog+ddrblock
LOG_STAGING_DIR         $INFORMIXDIR/tmp
CDR_LOG_STAGING_MAXSIZE 1MB
CDR_MAX_DYNAMIC_LOGS    2
DYNAMIC_LOGS            2
```

Related information

LOG_STAGING_DIR configuration parameter on page

Handle potential log wrapping on page 214

onmode -wf, -wm: Dynamically change certain configuration parameters on page

CDR_LOG_STAGING_MAXSIZE Configuration Parameter

Specifies the maximum amount of space that Enterprise Replication uses to stage compressed log files in the directory specified by the LOG_STAGING_DIR configuration parameter.

default value

0

onconfig.std value

CDR_LOG_STAGING_MAXSIZE 0

syntax

CDR_LOG_STAGING_MAXSIZE *sizeunit*

range of values

The range of values for *size* is:

- 0 = Default. Log staging is disabled.
- Positive integers = The maximum size of the stage log files.

The range of value for *unit* is:

- KB (default)
- MB
- GB
- TB

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

Use the CDR_LOG_STAGING_MAXSIZE configuration parameter to limit the size of the log staging directory. Logs are staged if all of the following conditions are true:

- Enterprise Replication detects a potential for log wrapping.
- The CDR_LOG_LAG_ACTION configuration parameter setting includes the logstage option.
- The LOG_STAGING_DIR configuration parameter is set.

  The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission.

When the contents of the staging directory reaches the maximum allowed size, Enterprise Replication stops staging log files. Enterprise Replication stops staging files only at a log file boundary; that is, a file is not staged in the middle of a log file.

Example

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by staging compressed logs until the staging directory reached 100 KB, you would set the following configuration parameters:

```
CDR_LOG_STAGING_MAXSIZE 100
CDR_LOG_LAG_ACTION      logstage
LOG_STAGING_DIR         $INFORMIXDIR/tmp
```

Related information

LOG_STAGING_DIR configuration parameter on page

Handle potential log wrapping on page 214

onmode -wf, -wm: Dynamically change certain configuration parameters on page

CDR_MAX_DYNAMIC_LOGS Configuration Parameter

Specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.

onconfig.std value

`0`

range of values

- `-1` add dynamic log files indefinitely
- `0` disable dynamic log addition
- `>0` number of dynamic logs that can be added

takes effect

when the database server is shut down and restarted, and the DYNAMIC_LOGS configuration parameter is set to 2 or when the **cdr change onconfig** command is used. For more information on the DYNAMIC_LOGS configuration parameter, see the *HCL® Informix® Administrator's Reference*.

The CDR_MAX_DYNAMIC_LOGS configuration parameter specifies the number of dynamic log file requests that Enterprise Replication can make in one server session. The DYNAMIC_LOGS configuration parameter must be set to 2.

Related information

Handle potential log wrapping on page 214

DYNAMIC_LOGS configuration parameter on page

CDR_MAX_FLUSH_SIZE configuration parameter

Specifies the maximum number of replicated transactions that are applied before the logs are flushed.

onconfig.std value

CDR_MAX_FLUSH_SIZE 50

default value if not present in the onconfig file

50

range of values

A positive integer that represents the maximum number of transactions to apply before the logs are flushed.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

By default, replication servers flush logs after 50 replicated transactions are applied, or after 5 seconds, whichever happens first.

If a replication server is a primary server for shared-disk secondary servers, you might want to reduce the replication latency. Set the CDR_MAX_FLUSH_SIZE configuration parameter to 1 to flush the logs after each replicated transaction.

Related information

onmode -wf, -wm: Dynamically change certain configuration parameters on page

Replication latency for secondary servers on page 92

CDR_MEM configuration parameter

The CDR_MEM configuration parameter is used to specify Enterprise Replication's method for memory-pool allocation.

onconfig.std value

CDR_MEM 0

values

`0`: Memory allocation from the generic pool is taken from the CDR pool. Memory allocation from the RQM pool is taken from the queue's memory pool.

`1`: Memory allocation pools are associated with specific CPU virtual processors. Enterprise Replication allocates memory to the CPU virtual processors based on which CPU virtual processor the cdr thread is executing on.

`2`: Memory allocation pools are associated with specific block sizes, so that all allocations from a pool are the same size, and the first free block that is found can be used.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

After you run the SQL administration API task() or admin() function with the `"onmode","-wf CDR_MEM=`*`value`*`"` or `"onmode","-wm CDR_MEM=`*`value`*`"` argument.

Usage

`CDR_MEM 0` is the traditional method of memory-allocation. Use this setting when resource allocation is more important than performance.

`CDR_MEM 1` prevents multiple threads from simultaneously accessing a memory pool. The performance of large-scale Enterprise Replication environments can improve, because memory allocation is done by multiple threads that are working in parallel.

`CDR_MEM 2` improves performance at the cost of increased memory usage. Memory allocation requests are increased to the closest fixed-block size, so that free memory blocks can be found faster. Memory pools are not associated with specific CPU virtual processors, so memory can be freed directly to the memory pool.

CDR_NIFCOMPRESS Configuration Parameter

Specifies the level of compression the database server uses before sending data from the source database server to the target database server.

onconfig.std value

`0`

range of values

- `-1` specifies no compression
- `0` specifies to compress only if the target server expects compression
- `1` - `9` specifies increasing levels of compression

takes effect

When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_NIFCOMPRESS (network interface compression) configuration parameter specifies the level of compression that the database server uses before sending data from the source database server to the target database server. Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data.

The values have the following meanings.

Value	Meaning
`-1`	The source database server never compresses the data, regardless of whether or not the target site uses compression.
0	The source database server compresses the data only if the target database server expects compressed data.
1	The database server performs a minimum amount of compression.
9	The database server performs the maximum possible compression.

When Enterprise Replication is defined between two database servers, the CDR_NIFCOMPRESS values of the two servers are compared and changed to the higher compression values.

The compression values determine how much memory can be used to store information while compressing, as follows:

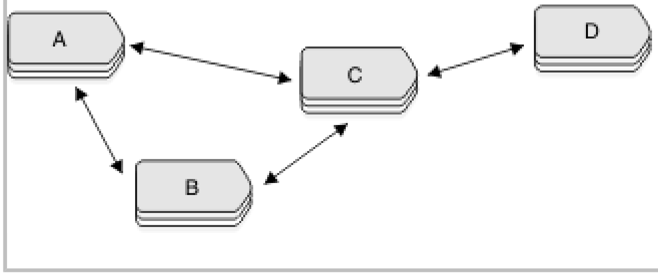
Example

```
0 = no additional memory
1 = 128k + 1k    = 129k
2 = 128k + 2k    = 130k
...
6 = 128k + 32k  = 160k
...
8 = 128k + 128k = 256k
9 = 128k + 256k = 384k
```

Higher levels of CDR_NIFCOMPRESS cause greater compression.

Different sites can have different levels. For example, Figure 27: Database Servers with Different Compression Levels on page 512 shows a set of three root servers connected with LAN and a nonroot server connected over a modem. The CDR_NIFCOMPRESS configuration parameter is set so that connections between A, B, and C use no compression. The connection from C to D uses level 6.

Figure 27. Database Servers with Different Compression Levels



Important: Do not disable NIF compression if the network link performs compression in hardware.

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers and SMX_COMPRESS shall be used to enable compression. For communicating with older server versions before 14.10xC6, Enterprise Replication still require configuring CDR_NIFCOMPRESS.

CDR_QDATA_SBSPACE Configuration Parameter

Specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data.

onconfig.std value

none

separators

comma

range of values

up to 128 characters for each sbspace name; up to 32 sbspace names. Use a comma to separate each name in the list. At least one sbspace name must be specified.

takes effect

when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_QDATA_SBSPACE configuration parameter specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data. Enterprise Replication creates one smart large object per transaction. The sbspaces must be used only for Enterprise Replication. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order.

Important: You must set the CDR_QDATA_SBSPACE configuration parameter and create the sbspaces specified by CDR_QDATA_SBSPACE before defining a server for replication. If the configuration parameter is not set in the `onconfig` file or the sbspace names specified by CDR_QDATA_SBSPACE are invalid, Enterprise Replication fails to define the server. For more information, see Row Data sbspaces on page 67 and Defining Replication Servers on page 93.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_QDATA_SBSPACE configuration parameter when defining a replication server. If the CDR_QDATA_SBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the cdr define server command automatically creates the necessary disk space and sets the configuration parameter to the appropriate value in memory and the `onconfig` file. For clusters, the cdr define command creates new sbspaces and sets the CDR_QDATA_SBSPACE configuration parameters in all secondary database servers, as well.

Note: A database server's storage pool must have 500 MB of free space for the sbspace. The sbspace must be comprised of chunks of size 100 MB or greater for the database server to use automatic storage provisioning.

Warning: Do not change the value of CDR_QDATA_SBSPACE while Enterprise Replication is running.

Related reference

cdr start sec2er on page 450

Related information

Monitoring Disk Usage for Send and Receive Queue Spool on page 215

Row Data sbspaces on page 67

CDR_QUEUEMEM Configuration Parameter

Specifies the maximum amount of memory that is used for the send and receive queues.

onconfig.std value

`131072`

units

kilobytes

range of values

From `500` through `4194304`

takes effect

When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_QUEUEMEM configuration parameter specifies the maximum amount of memory that the send and receive queues use for transaction headers and for transaction data. The total size of the transaction headers and transaction data in a send or receive queue could be up to twice the size of that value of CDR_QUEUEMEM. If your logical logs are large, the Enterprise Replication reads a large amount of data into queues in memory. You can use CDR_QUEUEMEM to limit the amount of memory devoted to the queues.

When you increase the value of CDR_QUEUEMEM, you reduce the number of elements that must be written to disk, which can eliminate I/O overhead. Therefore, if elements are frequently stored on disk, increase the value of CDR_QUEUEMEM. Conversely, if you set the value of CDR_QUEUEMEM too high, you might adversely impact the performance of your system. High values for CDR_QUEUEMEM also increase the time necessary for recovery. Tune the value of CDR_QUEUEMEM for the amount of memory available on your computer.

CDR_SERIAL Configuration Parameter

Enables control over generating values for serial columns in tables that are defined for replication.

onconfig.std value

CDR_SERIAL 0

range of values

`0` = Default. Disable control of serial column value generation.

delta,*offset* = Enable control of serial column value generation:

delta

A positive integer that sets the incremental size of the serial column values. This value must be the same on all replication servers and must be at least the number of expected servers in the Enterprise Replication domain.

offset

A positive integer that sets the offset of the serial value to be generated. This value must be different on all replication servers and must be between 0 and one less than the value of *delta*, inclusive.

takes effect

After you edit your `onconfig` file and restart the database server.

After you run the cdr change onconfig command.

The CDR_SERIAL configuration parameter controls generating values for SERIAL, SERIAL8, and BIGSERIAL columns in replicated tables so that no conflicting values are generated across multiple Enterprise Replication servers. You must set the CDR_SERIAL configuration parameter if the serial column is the replication key column and no other replication key column, such as a server ID, guarantees the uniqueness of the replication key. If the serial column is not the replication key, you can set the CDR_SERIAL configuration parameter to ensure that the serial values are unique across all servers. Only tables that are marked as the source of a replicate are controlled by the CDR_SERIAL configuration parameter settings.

For example, suppose that you have two primary servers, **g_usa** and **g_japan**, and one read-only target server, **g_italy**. You plan to add three more servers in the future. You might set CDR_SERIAL to the values shown in the following table.

Table 35. CDR_SERIAL Example Settings and Results

Server	Example CDR_SERIAL Value	Resulting Values for the Serial Column
g_usa	5,0	5, 10, 15, 20, 25, and so on
g_japan	5,1	1, 6, 11, 16, 21, 26, and so on
g_italy	0	no local inserts into the serial column

The following CDR_SERIAL settings are reserved for future servers:

- `5,2`
- `5,3`
- `5,4`

If you must add more servers than the *delta* value of CDR_SERIAL, you must reset CDR_SERIAL on all servers simultaneously and ensure that the serial values on the new servers are unique.

Related information

Serial data types and replication keys on page 29

CDR_SUPPRESS_ATSRISWARN Configuration Parameter

Specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files.

onconfig.std value

none

units

numbers or hyphen-separated ranges of numbers

separator

commas

takes effect

when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_SUPPRESS_ATSRISWARN configuration parameter specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files. For example, you can set CDR_SUPPRESS_ATSRISWARN to 2-5, 7 to suppress the generation of error and warning messages 2, 3, 4, 5, and 7. For a list of error and message numbers see Data sync warning and error messages on page 610.

CDR_TSINSTANCEID configuration parameter

Specifies how to generate unique identifiers for time series instances across replication servers. If a replicate includes a column with a **TimeSeries** column, the CDR_TSINSTANCEID configuration parameter must be set to a different value on every participating replication server before you create any time series instances.

onconfig.std value

CDR_TSINSTANCEID 0

range of values

0 = Default. Disable the replication of **TimeSeries** columns.

1 - 32768 = The number that is added to the time series instance identifiers.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

You set the value of CDR_TSINSTANCEID to a different value on each replication server that replicates a **TimeSeries** column. A time series instance identifier is automatically generated when you create a time series instance. The unique values of CDR_TSINSTANCEID configuration parameter on each replication server ensures that no time series instance identifiers overlap in the replication domain.

Related information

onmode -wf, -wm: Dynamically change certain configuration parameters on page

ENCRYPT_CDR Configuration Parameter

Use the ENCRYPT_CDR configuration parameter to set the level of encryption for Enterprise Replication.

onconfig.std value

ENCRYPT_CDR 0

values

0 = Default. Do not encrypt.

1 = Encrypt when possible. Encryption is used for Enterprise Replication transactions only when the database server being connected to also supports encryption.

2 = Always encrypt. Only connections to encrypted database servers are allowed.

takes effect

After you edit your `onconfig` file and restart the database server.

After you run the cdr change onconfig command.

Usage

If you enable encryption with the ENCRYPT_CDR configuration parameter, you must also set the ENCRYPT_MAC, ENCRYPT_MACFILE, ENCRYPT_SWITCH, and ENCRYPT_CIPHERS configuration parameter to configure encryption.

If you use both encryption and compression (by setting the CDR_NIFCOMPRESS configuration parameter), then compression occurs before encryption.

Note:

- From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers and ENCRYPT_SMX or onsocssl shall be used to enable encryption. For communicating with older server versions before 14.10xC6, Enterprise Replication still requires configuring ENCRYPT_CDR.
- Support for Informix encryption is removed starting Informix Server 14.10.xC9 . ENCRYPT_CDR is used in Informix encryption. Hence, ENCRYPT_CDR is not supported starting 14.10.xC9.

Related information

ENCRYPT_CIPHERS configuration parameter on page

ENCRYPT_MACFILE configuration parameter on page

ENCRYPT_SWITCH configuration parameter on page

ENCRYPT_MAC configuration parameter on page

GRIDCOPY_DIR Configuration Parameter

Specifies the default directory used by the ifx_grid_copy procedure.

onconfig.std value

$INFORMIXDIR

default value if not present in the `onconfig` file

$INFORMIXDIR

values

pathname = `$INFORMIXDIR` or a valid file path that is relative to `$INFORMIXDIR`.

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the onmode -wf command.

When you reset the value in memory by running the onmode -wm command.

The ifx_grid_copy() procedure copies files from a grid database server to the other nodes of the same grid. The GRIDCOPY_DIR value is the default directory of file paths in the ifx_grid_copy() command:

- On a database server running the ifx_grid_copy() procedure, the GRIDCOPY_DIR value and the ifx_grid_copy() command *source_path_and_filename* is the location from which files are copied.
- On a database server sharing the same grid with a node running the ifx_grid_copy() procedure, the GRIDCOPY_DIR value and the ifx_grid_copy() command's *target_path_and_filename* is the location to which files are copied. If *target_path_and_filename* is not specified as part of the ifx_grid_copy() command, the GRIDCOPY_DIR value and the ifx_grid_copy() command's *source_path_and_filename* is the location to which files are copied.

If a node directory specified by a GRIDCOPY_DIR value does not exist, the node directory is created by the ifx_grid_copy() procedure.

Example

Example

To specify `$INFORMIXDIR/usr/informix/copydir` as a node's default directory for ifx_grid_copy() procedure actions, set the following value in the `onconfig` file:

```
GRIDCOPY_DIR usr/informix/copydir
```

Related reference

ifx_grid_copy() procedure on page 531

Related information

Propagating external files through a grid on page 142

SHARD_ID configuration parameter

Sets the unique ID for a shard server in a shard cluster.

onconfig.std value

SHARD_ID 0

range of values

0 = Default. The database server cannot run parallel sharded queries.

1 - 65535 = The unique ID of the shard server.

takes effect

After you edit your `onconfig` file and restart the database server.

If the value is 0 or not set, you can set the value dynamically in your `onconfig` file by running the onmode -wf command.

You set the value of the SHARD_ID configuration parameter to a different number on each shard server in a shard cluster. If the value of the SHARD_ID configuration parameter is unset or set to 0 on all shard servers in the shard cluster, the shard cluster performs poorly. If the values of the SHARD_ID configuration parameter are not unique on all shard servers in a shard cluster, shard queries fail.

To reset the value if the SHARD_ID configuration parameter is set to a positive integer, edit the `onconfig` file and then restart the database server.

Related information

Creating a shard cluster on page 153

onmode -wf, -wm: Dynamically change certain configuration parameters on page

CDR_ALARMS Environment Variable

Enables Enterprise Replication event alarms.

default value

30-39,47,48,50,71,73-75

range of values

integers in these ranges: 30-39, 47-71, 73-75

separator

comma (,) to separate individual numbers or hyphen (-) to separate a range of numbers

takes effect

When the database server is shut down and restarted.

Set the CDR_ALARMS environment variable to the Enterprise Replication event alarms that you want to receive. Enterprise Replication event alarms that are not set by CDR_ALARMS are disabled.

Use the CDR_ENV configuration parameter to set this environment variable in the `onconfig` file.

Related information

Enabling or Disabling Enterprise Replication Event Alarms on page 244

CDR_ATSRISNAME_DELIM Environment Variable

Specifies the delimiter to use to separate the parts of the time portion of ATS and RIS file names that are in text format.

default value

On UNIX™: a colon (`:`)

On Windows™: a period (`.`)

range of values

a single character

takes effect

when Enterprise Replication is initialized

ATS and RIS files in XML format always use a period (.) as the delimiter.

For example, the default file name for an ATS file in text format on UNIX™ might look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6**. If CDR_ATSRISNAME_DELIM is set to a period (.), then the same file name would look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6**.

Related information

ATS and RIS File Names on page 201

CDR_DISABLE_SPOOL Environment Variable

Controls the generation of ATS and RIS files.

default value

`0`

range of values

`0` Allow ATS and RIS file generation

`1` Prevent ATS and RIS file generation

takes effect

when Enterprise Replication is initialized

The CDR_DISABLE_SPOOL environment variable controls whether ATS and RIS files are generated. Set CDR_DISABLE_SPOOL to `1` if you do not want ATS or RIS files to be generated under any circumstances.

Related information

Failed Transaction (ATS and RIS) Files on page 199

Disabling ATS and RIS File Generation on page 212

CDR_LOGDELTA Environment Variable

Determines when the send and receive queues are spooled to disk as a percentage of the logical log size.

default value

`30`

range of values

positive numbers

takes effect

when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_LOGDELTA environment variable determines when the send and receive queues are spooled to disk as a percentage of the logical log size. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter on page 502.

Important: Do not use the CDR_LOGDELTA environment variable unless instructed to do so by Technical Support.

CDR_PERFLOG Environment Variable

Enables queue tracing.

default value

`0`

range of values

positive number

takes effect

when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_PERFLOG environment variable enables queue tracing. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter on page 502.

Important: Do not use the CDR_PERFLOG environment variable unless instructed to do so by Technical Support.

CDR_RMSCALEFACT Environment Variable

Sets the number of data sync threads started for each CPU VP.

default value

`4`

range of values

positive number

takes effect

when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_RMSCALEFACT environment variable sets the number of data sync threads started for each CPU VP. Specifying a large number of threads can result in wasted resources. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter on page 502.

Important: Do not use the CDR_RMSCALEFACT environment variable unless instructed to do so by Support.

CDR_ROUTER Environment Variable

Disables intermediate acknowledgments of transactions in the hierarchical topologies.

default value

0

range of values

any number

takes effect

when Enterprise Replication is initialized or immediately after the cdr change onconfig command is used

When set to 1, the CDR_ROUTER environment variable disables intermediate acknowledgments of transactions in hierarchical topologies. The normal behavior for intermediate servers is to send acknowledgments if they receive an acknowledgment from the next server in the replication tree (can be a leaf server) or if the transaction is stored in the local queue. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter on page 502.

If CDR_ROUTER is set at the hub server, an acknowledgment will be sent only if the hub server receives acknowledgment from all of its leaf servers. Transactions will not be acknowledged even if they are stored in the local queue of the hub server.

If CDR_ROUTER is not set at hub server, the hub server will send an acknowledgment if the transaction is stored in the local queue at the hub server or if the hub server received acknowledgment from all of its leaf servers.

Important: Do not use the CDR_ROUTER environment variable unless instructed to do so by Technical Support.

CDRSITES_10X Environment Variable

Works around a malfunction in version reporting for fix pack versions of 10.00 servers.

units

cdrIDs, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i** ***=unique_ID***)

range of values

positive numbers

takes effect

when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_10X** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve Versions 10.00.xC1 or 10.00.xC3 servers, the NIF does not properly report its version when it responds to a new server with a fix pack version of 10.00.xC4 or later. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server.

To prevent this malfunction, if you have Version 10.00.xC1 or 10.00.xC3 servers in your Enterprise Replication environment, set the **CDRSITES_10X** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_10X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_10X** as a standard environment variable.

The ***cdrID*** is the unique identifier for the database server in the Options field of the SQLHOSTS file (**i =** ***unique_ID***).

For example, suppose that you have 5 database servers, Version 10.00.xC1, whose ***cdrID*** values range from 2 through 10 (***cdrID*** = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.00.xC4, you must set the **CDRSITES_10X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.00.xC4 database server online:

```
CDR_ENV CDRSITES_10x=2,3,9,10
```

CDRSITES_731 Environment Variable

Works around a malfunction in version reporting for post-7.3x, 7.20x, or 7.24x version servers.

units

cdrIDs, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i** **=*****unique_ID***)

range of values

positive numbers

takes effect

when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_731** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve post 7.3x, 7.20x, or 7.24x servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0, 9.40, or 9.30 server tries to synchronize with a 7.3x, 7.20x, or 7.24x server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 7.3x, 7.20x, or 7.24x servers in your Enterprise Replication environment, set the **CDRSITES_731** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_731** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_731** as a standard environment variable.

For example, suppose that you have 5 database servers, Version 7x servers whose ***cdrID*** values range from 1 through 7 (***cdrID*** = 1, 4, 5, 6, and 7).

If you upgrade database server ***cdrID*** 6 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_731** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_731=1,4,5,7
```

CDRSITES_92X Environment Variable

Works around a malfunction in version reporting for 9.21 or 9.20 servers.

units

cdrIDs, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values

positive numbers

takes effect

when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_92X** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve 9.21 or 9.20 servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0/9.40/9.30 server tries to synchronize with a 9.21 or 9.20 server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 9.21 or 9.20 servers in your Enterprise Replication environment, set the **CDRSITES_92X** environment variable

Note: You can only set the **CDRSITES_92X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_92X** as a standard environment variable.

For example, suppose that you have 5 database servers, Version 9.21 or 9.20 whose ***cdrID*** values range from 2 through 10 *(cdrIDs* = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_92X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_92x=2,3,9,10
```

Grid routines

Grid routines are used to create and maintain the grid and to administer servers in the grid by propagating commands from a source server to all other servers in the grid.

ifx_get_erstate() function

The ifx_get_erstate() function indicates whether replication is enabled for the transaction in which it is run.

Syntax

```
EXECUTE FUNCTIONifx_get_erstate( )INTOdata_var;
```

Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the ifx_get_erstate() function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the ifx_set_erstate() procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

Example

The following example obtains the replication state and stores it in the **curstate** variable:

```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related reference

ifx_set_erstate() procedure on page 546

Related information

Enabling replication within a grid transaction on page 137

Recapture replicated transactions on page 196

ifx_grid_connect() procedure

The ifx_grid_connect() procedure opens a connection to the grid. Through an ifx_grid_connect() procedure, you can run routines and data definition language (DDL) commands on a source server, and then propagate the routines or commands to the other grid servers.

Syntax

```
EXECUTE PROCEDUREifx_grid_connect( 'grid_name' {[ ,'tag ' ][ ,ER_enable ] | ,'tag' ,defer } );
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
ER_enable	Enable or disable the creation of a replicate and replicate set and starting replication for any tables that are created while the connection to the grid is open. Optionally suppress any errors that might be raised when the procedure is run.	Valid values are: • 0 = Default. Enterprise Replication is disabled. • 1 = Enterprise Replication is enabled. • 2 = Enterprise Replication is disabled and errors are suppressed. • 3 = Enterprise Replication is enabled and errors are suppressed.
defer	Run DDL statements on the local server but delay the propagation of the statements to other servers in the grid. Optionally enable the creation of a replicate and replicate set and starting replication.	Valid values are: • 4 = Defer the propagation of DDL statements. • 5 = Defer the propagation of DDL statements and enable Enterprise Replication. Use this value when you run DDL statements on existing replicated tables.

Element	Purpose	Restrictions
tag	A character string to identify grid operations.	Must be unique among grid sessions with deferred DDL statements that are outstanding.

Usage

Use the ifx_grid_connect() procedure start a grid connection. All DDL SQL statements and routines that you run in the grid connect are propagated to all the servers in the grid. Use the ifx_grid_disconnect() procedure to close the grid connect and disable grid propagation. If the databases on your replication servers have different schemas or data, a DDL statement that is run through a grid might have different results on each server. In a replication system, when you run a statement locally, the results are replicated to the other replication servers. When you run a statement through a grid, that statement is simultaneously run on each server.

You must run this routine as an authorized user on an authorized server, as specified by the cdr enable grid command.

You must connect to a database before you run the ifx_grid_connect() procedure. If you are planning to create a database, you can connect to the **sysmaster** database.

If you enable Enterprise Replication, when you create a table through the grid, a replicate is created that contains the newly created table with all the servers in the grid as participants. The replicate belongs to a replicate set that has the same name as the grid. When you create a replicated table through the grid, the ERKEY shadow columns are added automatically.

If you run the ifx_grid_connect() procedure automatically as part of the sysdbopen() procedure, set the *ER_enable* argument to 2 or 3 to suppresses errors that might prevent the session from accessing the database.

You can defer the propagation of DDL statements to other servers in the grid by setting the *defer* argument. The DDL statements are queued for propagation but not sent to other grid servers until you run the ifx_grid_release() function.

You cannot perform the following actions in the context a grid connection:

- Propagate data manipulation language statements through a grid.
- Replicate a database object that exists on a server in the grid.
- Use the `@servername` syntax while connected to the grid.
- Drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.
- Renaming a replicated database. You must manually rename the database on each participant server.

Example

Example 1: Create a table

In the following example, a grid connection is opened that enables the propagation of only DDL statements, a table is created on all servers in the grid, and then the grid connection is closed:

```
database sales;
```

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate date,
    offer_enddate date,
    offer_rules  lvarchar);

EXECUTE PROCEDURE ifx_grid_disconnect();
```

In this example, the data in the **special_offers** table is not replicated.

Example

Example 2: Create a replicated table

In the following example, a grid connection is opened that enables the propagation of DDL statements and the replication of data, a table is created on all servers in the grid, and then the grid connection is closed:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
   offer_description varchar(255),
    offer_startdate date,
    offer_enddate date,
    offer_rules  lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

A replicate for the **special_offers** table is created with timestamp conflict resolution and replication of the data in the table is started.

Example

Example 3: Alter a replicated table to add a column

The following example alters the **special_offers** table to add a column and remasters the replicate on all participants that are members of the grid:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
   offer_exceptions  varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Example

Example 4: Alter a replicated table to add a column that is not replicated

The following example alters the **special_offers** table to add a column whose data is not replicated:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 0);

ALTER TABLE special_offers ADD (
   local_promotions  varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

The column **local_promotions** is added to the **special_offers** table on all grid servers, but the data in the **local_promotions** column is not replicated.

Example

Example 5: Defer propagation of a DDL statement

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
   local_restrictions  varchar(255));
```

The column **local_restrictions** is added to the **special_offers** table on the local server only. The ALTER operation is queued for propagation to the other grid servers.

Related reference

ifx_grid_disconnect() procedure on page 532

cdr enable grid on page 386

ifx_grid_release() function on page 537

ifx_grid_remove() function on page 539

cdr delete grid on page 372

Related information

Propagating database object changes on page 141

Grid queries on page 145

Creating replicated tables through a grid on page 136

Adding an existing replicate to a grid replicate set by altering a table on page 135

ifx_grid_copy() procedure

The ifx_grid_copy() procedure copies non-database, external files from a grid database server to other nodes in the same grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_copy( 'grid_name', 'source_path_and_filename' [ ,'target_path_and_filename ' ] );
```

Element	Purpose	Restriction
grid_name	The name of the database server's grid.	
source_path_and_filename	The file path, relative to the value of the GRIDCOPY_DIR configuration parameter on the source database server, and name of the file you want to send to the other nodes of the grid.	The file must be located relative to the directory specified by the GRIDCOPY_DIR configuration parameter on the source server. By default, the GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR.
target_path_and_filename	The file path, relative to values of the GRIDCOPY_DIR configuration parameter on each other node of a grid, and the name each file will have on the other nodes of the grid.	If you do not specify *target_path_and_filename*, the file is copied to *source_path_and_filename* on each node, relative to the GRIDCOPY_DIR configuration parameter value on that node.

Usage

The ifx_grid_copy() procedure copies a file, along with the file's permissions, group, and owner values, from a directory on a grid server to specified destinations on each other node that is currently part of the grid. Group and owner values, rather than group ID and user ID values, are copied because group IDs and user IDs can have different values on different servers. If the file's group or owner values have not been defined on a node receiving a copied file, the copy fails on that node.

You can run this procedure only on an authorized database server and as an authorized user, as specified by the cdr grid enable command.

Only nodes that are members of a grid at the time the ifx_grid_copy() is run receive the copied file. Grid nodes added after ifx_grid_copy() procedures complete are not updated with files previously copied to other nodes.

Nonexistent directories that are specified by the *source_path_and_filename* or *target_path_and_filename* values of the ifx_grid_copy() command are created during the ifx_grid_copy() procedure.

Wildcard characters in file names are not supported.

Example

Example 1: Copying a file to the same location on database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to `$INFORMIXDIR/tmp` on all database servers of a grid named **grid1**. The following command copies the file `script1.exe` from `$INFORMIXDIR/tmp/bin` on the source database server to `$INFORMIXDIR/tmp/bin` on all other nodes of **grid1**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid1", "bin/script1.exe");
```

Example

Example 2: Copying a file to the same relative locations on other database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to `$INFORMIXDIR/tmp` on the source database server and `$INFORMIXDIR/copies` on each other node of a grid named **grid2**. The following command copies the file `script2.exe` from `$INFORMIXDIR/tmp/bin` on the source database server to `$INFORMIXDIR/copies/bin` on all other nodes of **grid2**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid2", "bin/script2.exe");
```

Example

Example 3: Copying a file to different relative locations on the other database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to `$INFORMIXDIR/tmp` on all database servers of a grid named **grid3**. The following command copies the file `script3.exe` from `$INFORMIXDIR/tmp/bin` on the source server to `$INFORMIXDIR/tmp/copies` on all other nodes of **grid3**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid3", "bin/script3.exe", "copies/script3.exe");
```

Example

Example 4: Changing the name of file copied throughout a grid

The GRIDCOPY_DIR configuration parameter is set to `$INFORMIXDIR/tmp` on all database servers of a grid named **grid4**. The following command copies the file `script4.exe` in `$INFORMIXDIR/tmp` on the source server to `$INFORMIXDIR/tmp` on all other nodes of **grid4**. The copied file is named `script4_copy.exe` on the other nodes of **grid4**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid4", "script4.exe", "script4_copy.exe");
```

Related reference

GRIDCOPY_DIR Configuration Parameter on page 518

Related information

Propagating external files through a grid on page 142

ifx_grid_disconnect() procedure

The ifx_grid_disconnect() procedure closes a connection to the grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_disconnect ();
```

Usage

Use the ifx_grid_disconnect() procedure to disable the propagation of DDL statements and commands to servers in the grid, which was enabled by the ifx_grid_connect() procedure. If you do not use the ifx_grid_disconnect() procedure, propagation through the grid is stopped when the database is closed or the connection is closed.

You must run this routine as an authorized user on an authorized grid server, as specified by the cdr grid enable command.

Example

Example

The following example shows how to close a connection to the grid after opening a connection:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related reference

ifx_grid_connect() procedure on page 527

Related information

Propagating database object changes on page 141

ifx_grid_execute() procedure

The ifx_grid_execute() procedure propagates the execution of a routine or data manipulation language (DML) SQL statement to all servers in the grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_execute( 'grid_name' , 'statement_text ' [ , 'tag ' ] );
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
statement_text	The text format of the routine or SQL statement to be run.	Bound data items cannot be included in procedure text.
tag	A character string to identify grid operations.	

Usage

Use the ifx_grid_execute() procedure to run a routine or DML SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the routine, if any, is not returned to the client application. The results of routines or statements that are performed within the context of the ifx_grid_execute() procedure are not replicated.

The ifx_grid_execute() procedure effectively runs routines and statements with a BEGIN WORK WITHOUT REPLICATION statement. Do not use the ifx_grid_execute() procedure to populate tables that are already involved in replication. Although you can use the ifx_grid_execute() procedure to run a DML statement, for example, to delete many rows from a table, in general use Enterprise Replication to replicate changes to replicated data. You can run DML statements on any type of table, including raw tables, virtual tables, and external tables.

You cannot run the ifx_grid_execute() procedure from within a transaction. When you run SQL administration API commands from the ifx_grid_execute() procedure, you must use double quotation marks around the SQL administration API function arguments and single quotation marks around the ifx_grid_execute() procedure arguments.

You must run this routine as an authorized user on an authorized server, as specified by the cdr grid enable command.

Example

Example

The following example, run from the **sysadmin** database, uses an SQL administration API command to create a dbspace on every server in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
  'admin("create dbspace", "dbspace3",
  "$INFORMIXDIR/WORK/dbspace3", "500 M")');
```

The following example drops the logical logs from the chunk number 3 from all the servers in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1', 'SELECT task("drop log", number) FROM
     sysmaster:syslogfil where chunk = 3;');
```

Related reference

ifx_grid_procedure() procedure on page 535

ifx_grid_function() function on page 534

Related information

Administering servers in the grid with the SQL administration API on page 139

Propagating updates to data on page 138

ifx_grid_function() function

The ifx_grid_function() function propagates the execution of a function to all servers in the grid.

Syntax

```
EXECUTE FUNCTIONifx_grid_function( 'grid_name' ,'function_text ' [ ,'tag ' ] );
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.

Element	Purpose	Restrictions
function_text	The text format of the function to be run.	
tag	A character string to identify grid operations.	

Usage

Use the ifx_grid_function() function to run a function or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the function is returned to the client application as an LVARCHAR data type with comma-delimited text. You can also view the output with the cdr list grid command with the --verbose option. You cannot run the ifx_grid_function() function from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the cdr grid enable command.

Example

Example

The following example runs a function named load_function():

```
EXECUTE FUNCTION ifx_grid_function('grid1', 'load_function(2000)');
```

Related reference

ifx_grid_execute() procedure on page 533

Related information

Administering servers in the grid with the SQL administration API on page 139

ifx_grid_procedure() procedure

The ifx_grid_procedure() procedure propagates the execution of a procedure to all servers in the grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_procedure ( 'grid_name' ,'procedure_text ' [ ,'tag ' ] );
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	
procedure_text	The text format of the procedure to be run.	Bound data items cannot be included.	
tag	A character string to identify grid operations.		

Usage

Use the ifx_grid_procedure() procedure to run a procedure or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. You cannot run the ifx_grid_procedure() procedure from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the cdr grid enable command.

Example

Example

The following example runs a procedure named myloadprocedure():

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1',
 'myloadprocedure(2000)', 'mytag');
```

Related reference

ifx_grid_execute() procedure on page 533

ifx_grid_redo() procedure

The ifx_grid_redo() procedure reruns commands that were run through the grid and failed on one or more servers in the grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_redo( 'grid_name' [ , 'source_server ' [ , 'target_server ' [ , 'tag ' [ , 'command_ID ' [ , 'force
' ] ] ] ] ] );
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to rerun on the grid.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server from which the routine was run.	Can be NULL.
tag	A character string identifying the grid operations to rerun.	Must be an existing tag. Can be NULL.
target_server	The replication server on which to rerun the routine.	Can be NULL.

Usage

Commands that you run through the grid might fail on one or more servers in the grid. Use the ifx_grid_redo() procedure to rerun commands that failed. For example, if you create a fragmented table through the grid and one of the grid servers does not have one of the dbspaces into which the table is fragmented, the command fails on that server. After you add the required dbspace to the server, run the ifx_grid_redo() procedure to create the fragmented table on that server.

You can specify from which source server the commands were run, the command ID, the target server on which the commands failed, or the identifying tag for commands that failed.

Use the **force** argument to rerun commands that succeeded.

You must run this routine as an authorized user on an authorized server, as specified by the cdr grid enable command.

Example

Example

The following example reruns failed commands on every server in the grid on which those commands failed:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1');
```

The following example reruns the command with the ID of 21 that originated server **cdr1** on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example reruns all commands that failed on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', NULL, 'cdr4');
```

Related information

Rerunning failed grid routines on page 143

ifx_grid_release() function

The ifx_grid_release() function propagates deferred DDL statements that were run on the local grid server, but deferred from running on the other grid servers.

Syntax

```
EXECUTE FUNCTIONifx_grid_release( 'grid_name', 'tag');
```

Element	Purpose	Restrictions
grid_name	The name of the grid that a DDL statement is queued to propagates across	Must be the name of an existing grid.

Element	Purpose	Restrictions
tag	A character string to identify grid operations.	Must be the same value as the *tag* argument that was included in the ifx_grid_connect() procedure with the *defer* argument.

Usage

If you deferred the propagation of DDL statements by running the ifx_grid_connect() procedure with the *defer* argument, DDL statements are run on the local server but deferred from propagating across the grid. Run the ifx_grid_release() function to propagate the DDL statements to the other grid servers. You can run the ifx_grid_release() command at any time after the grid session in which the statements were deferred.

You must run this routine as an authorized user on an authorized server, as specified by the cdr enable grid command.

Returns

The number of DDL statements that are released.

Example

Example

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
   local_restrictions  varchar(255));
```

The following statement releases the queued ALTER operation:

```
EXECUTE FUNCTION ifx_grid_release('grid1','tag1');
```

The **local_restrictions** column is added to the **special_offers** table on the other grid servers.

Related reference

ifx_grid_connect() procedure on page 527

ifx_grid_remove() function on page 539

Related information

Grid queries on page 145

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID clause on page

ifx_grid_remove() function

The ifx_grid_remove() function removes any DDL statements that are deferred from propagation to grid servers.

Syntax

```
EXECUTE FUNCTIONifx_grid_remove( 'grid_name', 'tag');
```

Element	Purpose	Restrictions
grid_name	The name of the grid that has a deferred DDL statement.	Must be the name of an existing grid.
tag	A character string to identify grid operations.	Must be the same value as the *tag* argument that was included in the ifx_grid_connect() procedure with the *defer* argument.

Usage

If you deferred the propagation of DDL statements by running the ifx_grid_connect() procedure with the *defer* argument, DDL statements are run on the local server but deferred from propagating across the grid. If you decide to not propagate the deferred DDL statements, run the ifx_grid_remove() function to remove the deferred DDL statement. You can run the ifx_grid_remove() command at any time after the grid session in which the statements were deferred. The ifx_grid_remove() command does not roll back the DDL statements on the local server.

You must run this routine as an authorized user on an authorized server, as specified by the cdr enable grid command.

Returns

The number of DDL statements that are removed.

Example

Example

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
   local_restrictions  varchar(255));
```

The following statement removes the queued ALTER operation:

```
EXECUTE FUNCTION ifx_grid_remove('grid1','tag1');
```

The **local_restrictions** column remains in the **special_offers** table on the local server but is not added to the special_offers tables on the other grid servers.

Related reference

ifx_grid_connect() procedure on page 527

ifx_grid_release() function on page 537

Related information

Grid queries on page 145

SELECT_GRID session environment option on page

SELECT_GRID_ALL session environment option on page

GRID clause on page

ifx_grid_purge() procedure

The ifx_grid_purge() procedure deletes metadata about commands that have been run through the grid.

Syntax

```
EXECUTE PROCEDUREifx_grid_purge( 'grid_name' [ ,'source_server ' [ ,'target_server ' [ ,'tag ' [ ,'command_ID ' [ ,'force
' ]]]]] );
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to purge.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server on which the routine originated.	Can be NULL.
tag	A character string identifying the grid operations to purge.	Must be an existing tag. Can be NULL.
target_server	The replication server on which the routine was run.	Can be NULL.

Usage

Use the ifx_grid_purge() procedure to delete the history of commands successfully run from the grid. Accumulated command history can significantly increase the size of the **syscdr** database.

Use the **force** argument to delete the history of all commands, including those that failed.

You must run this routine as an authorized user on an authorized server, as specified by the cdr grid enable command.

Example

Example

The following example deletes the history for commands that ran successfully:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1');
```

The following example deletes the history for commands, including those that failed:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, NULL, NULL, NULL, 'force');
```

The following example deletes the command history with the ID of 21 that originated server **cdr1** and ran on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example deletes all commands that ran successfully on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, 'cdr4');
```

Related information

Grid maintenance on page 130

ifx_gridquery_skipped_nodes() function

The ifx_gridquery_skipped_nodes() function returns the name of a server that was unavailable during a grid or shard query.

Syntax

```
EXECUTE FUNCTIONifx_gridquery_skipped_nodes ();
```

Usage

If you set the GRID_NODE_SKIP option of the SET ENVIRONMENT statement to `’on’`, any servers that are unavailable when a grid or shard query is run are skipped. Run the ifx_gridquery_skipped_nodes() function to return the name of a skipped server.

Use the ifx_gridquery_skipped_nodes() function with the ifx_gridquery_skipped_node_count() function. Run the ifx_gridquery_skipped_node_count() function to determine how many servers were skipped, and then run the ifx_gridquery_skipped_nodes() function the same number of times as the number of skipped servers.

Return value

An LVARCHAR string that supplies the name of a skipped server.

A return value of `0` indicates no skipped servers.

Example

Example

The following statement returns the number of skipped nodes in the grid or shard query:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();

2
```

The following statements return the names of the two skipped nodes:

```
EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

server1

EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

server2
```

Related reference

ifx_gridquery_skipped_node_count() function on page 542

Related information

Examples of grid queries on page 149

Grid queries on page 145

GRID_NODE_SKIP session environment option on page

GRID clause on page

ifx_gridquery_skipped_node_count() function

The ifx_gridquery_skipped_node_count() function returns the number of servers that were unavailable during a grid or shard query.

Syntax

```
EXECUTE FUNCTIONifx_gridquery_skipped_node_count ();
```

Usage

If you set the GRID_NODE_SKIP option of the SET ENVIRONMENT statement to `'on'`, any servers that are unavailable when a grid or shard query is run are skipped. Run the ifx_gridquery_skipped_node_count() function to return the number of a skipped servers.

Use the ifx_gridquery_skipped_node_count() function with the ifx_gridquery_skipped_nodes() function. Run the ifx_gridquery_skipped_node_count() function to determine how many servers were skipped, and then run the ifx_gridquery_skipped_nodes() function the same number of times as the number of skipped servers.

After a grid or shard query is run, the next SELECT statement that is run resets skipped node count, so ifx_gridquery_skipped_node_count() must be used in an EXECUTE FUNCTION statement.

Return value

An integer that indicates the number of skipped servers.

0 indicates that no grid or shard servers were skipped.

Example

Example

The following statement returns the number of skipped nodes in the grid or shard query:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();

2
```

Related reference

ifx_gridquery_skipped_nodes() function on page 541

Related information

Examples of grid queries on page 149

Grid queries on page 145

GRID_NODE_SKIP session environment option on page

GRID clause on page

ifx_node_id() function

The ifx_node_id() function returns the ID of the grid server on which the function is run.

Syntax

```
EXECUTE FUNCTIONifx_node_id( );
```

Usage

Use the ifx_node_id() function in the context of a grid query to return the ID of each server on which the grid query is run. Include the ifx_node_id() function in the SELECT statement of a grid query. The server ID is returned as a result column to identify the origin of the other results of the query.

If you run the ifx_node_id() function outside of the context of a grid query, the function returns the ID of the local server, unless you prefix the remote database and server name, for example: db@serv4:ifx_node_id().

Example

Example

The following grid query selects the server ID and the total sales from a grid named **SE_USA** and groups the results by the server ID:

```
SELECT ifx_node_id() AS ifx_node_id, sum(amt) AS total_sales
FROM sales GRID ALL 'SE_USA'
GROUP BY ifx_node_id;
```

```
ifx_node_id      total_sales


          1         $2100.00
          2         $2160.00
          3         $2000.00
          4         $2040.00
```

Related reference

cdr define region on page 341

cdr delete region on page 373

ifx_node_name() function on page 544

Related information

Examples of grid queries on page 149

Grid queries on page 145

GRID clause on page

ifx_node_name() function

The ifx_node_name() function returns the name of the grid server on which the function is run.

Syntax

```
EXECUTE FUNCTIONifx_node_name( );
```

Usage

Use the ifx_node_name() function in the context of a grid query to return the name of each server on which the grid query is run. Include the ifx_node_name() function in the SELECT statement of a grid query. The server name is returned as a result column to identify the origin of the other results of the query.

If you run the ifx_node_name() function outside of the context of a grid query, the function returns the name of the local server, unless you prefix the remote database and server name, for example: db@serv4:ifx_node_name().

Example

Example

The following grid query selects the server name and the total sales from a grid named **SE_USA** and groups the results by the server name:

```
SELECT ifx_node_name() AS node, sum(amt) AS total_sales
FROM sales GRID ALL 'SE_USA'
GROUP BY node;


node         Atlanta
total_sales  $2100.00
```

```
node         Birmingham
total_sales  $2160.00

node         Nashville
total_sales  $2000.00

node         Jacksonville
total_sales  $2040.00
```

Related reference

cdr define region on page 341

cdr delete region on page 373

ifx_node_id() function on page 543

Related information

Examples of grid queries on page 149

Grid queries on page 145

GRID clause on page

Enterprise Replication routines

Enterprise Replication routines used to control if a replicated transaction is recaptured.

ifx_get_erstate() function

The ifx_get_erstate() function indicates whether replication is enabled for the transaction in which it is run.

Syntax

```
EXECUTE FUNCTIONifx_get_erstate( )INTOdata_var;
```

Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the ifx_get_erstate() function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the ifx_set_erstate() procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

Example

The following example obtains the replication state and stores it in the **curstate** variable:

```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related reference

ifx_set_erstate() procedure on page 546

Related information

Enabling replication within a grid transaction on page 137

Recapture replicated transactions on page 196

ifx_set_erstate() procedure

The ifx_set_erstate() procedure controls whether database operations are replicated.

Syntax

```
EXECUTE PROCEDUREifx_set_erstate( {1 | 0 | 'on' | 'off' | data_var } );
```

Element	Purpose	Restriction
data_var	Variable holding the value that a function returned	

Usage

Use the ifx_set_erstate() procedure to enable or disable replication during a transaction. During normally replicated transactions, use the ifx_set_erstate() procedure to enable the recapture of a transaction after it has been replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Replication can only be enabled on tables that are participants in an existing replicate. To enable replication, set the ifx_set_erstate() procedure to `1` or `'on'`. To disable replication, set the ifx_set_erstate() procedure to `0` or `'off'`. To set replication to a previous state that was saved by the ifx_get_erstate() function, set the ifx_set_erstate() procedure to the name of the variable returned by the ifx_get_erstate() function.

Example

Example

The following example enables replication in the transaction:

```
EXECUTE PROCEDURE ifx_set_erstate(1);
```

The following example resets the replication state to a previous state that was saved by the ifx_get_erstate() function in the **curstate** variable:

```
EXECUTE PROCEDURE ifx_set_erstate(curstate);
```

Related reference

ifx_get_erstate() function on page 526

Related information

Enabling replication within a grid transaction on page 137

Recapture replicated transactions on page 196

onstat -g commands for Enterprise Replication

You can monitor and debug Enterprise Replication activity using onstat -g commands.

The onstat utility reads shared-memory structures and provides statistics about the database server that are accurate at the instant that the command executes. The system-monitoring interface (SMI) also provides information about the database server. For general information about onstat and SMI, refer to the *HCL® Informix® Administrator's Reference*. For information on SMI tables specific to Enterprise Replication, see SMI Tables for Enterprise Replication Reference on page 574.

Related information

onstat -k command: Print active lock information on page

onstat -g ath command: Print information about all threads on page

Monitor and troubleshooting Enterprise Replication on page 197

Threads shown by the onstat -g ath command

The threads that Enterprise Replication uses are shown by the onstat -g ath command.

The following table summarizes the threads that Enterprise Replication uses. You can use this information about threads when you evaluate memory use.

Table 36. Enterprise Replication threads

Number of Threads	Thread Name	Thread Description
1	ddr_snoopy	Performs physical I/O from logical log, verifies potential replication, and sends applicable log-record entries to Enterprise Replication.
1	preDDR	Runs during queue recovery to monitor the log and blocks user transactions if the log position advances too far before replication resumes.
1	CDRGfan	Receives log entries and passes entries to evaluator thread
n	CDRGeval*n*	Evaluates log entry to determine whether to replicated it. The value of *n* is the number of evaluator threads specified by the CDR_EVALTHREADS

Table 36. Enterprise Replication threads (continued)

Number of Threads	Thread Name	Thread Description
		configuration parameter. This thread also compresses committed transactions and queues completed replication messages.
1 per large transaction	CDRPager	Performs the physical I/O for the temporary smart large object that holds paged transaction records. Grouper paging is activated for a transaction when its size is 10 percent of the value of the SHMVIRTSIZE or CDR_QUEUEMEM configuration parameters or when it includes more than 100,000 records.
1	CDRCparse	Parses all SQL statements for replicate definitions.
1 per connection	CDRNsT*n*CDRNsA*n*	Sending thread for site.
1 per connection	CDRNr*n*	Receiving thread for site.
2...*n*	CDRACK_*n*	Accepts acknowledgments from site. At least 2, up to a maximum of the number of active connections.
# CPUs...	CDRD_*n*	Replays transaction on the target system (data sync thread). At least one thread is created for each CPU virtual processor (VP). The maximum number of threads is 4*(number of CPU VPs).
1	CDRSchedMgr	Schedules internal Enterprise Replication events.
0 or 1	CDRM_Monitor	Monitors and adjusts data sync performance for optimum performance (on the target).
0 or 1	CDRDTCleaner	Deletes rows from the deleted rows shadow table when they are no longer needed.

Related information

onstat -g ath command: Print information about all threads on page

onstat -g cat: Print ER global catalog information

Prints information from the Enterprise Replication global catalog.

```
onstat -g cat [{replname | full | repls | servers}]
```

Modifier	Description
replname	The name of a replicate

Usage

The global catalog contains a summary of information about the defined servers, replicates, and replicate sets on each of the servers within the domain. If a replicated table is undergoing an alter operation, the onstat -g cat command shows that it is in alter mode. For example, use this command to determine:

- How many servers and how many replicates are configured
- Which table matches a given replicate
- Whether a server is a root or leaf server
- The current bitmap mask for the specified server. You can use the bitmap mask with the output from the onstat -g rqm command to determine which server Enterprise Replication is waiting on for an acknowledgment.

You can set the scope of the output by specifying one of the following options to onstat -g cat:

- full: (Default) Prints expanded information for both replicate servers and replicates.
- *replname*: Prints information about the specified replicate only.
- repls: Prints information about replicates only.
- servers: Prints information about servers only.

Example

This sample output from the onstat -g cat repls command shows that the table **tab** is in alter mode. The replicate **rep1** is defined on this table, its replicate ID is 6553601.

```
GLOBAL-CATALOG CACHE STATISTICS
REPLICATES
-------------------
Parsed statements:
    Id 6553601 table tab
    Id 6553602 table tab12
Inuse databases: test(2)
  Name: rep1, Id: 6553601 State: ACTIVE Flags: 0x800000 ALTERMODE
          use 0 lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
          sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 12
    Column Name Verifcation: ON
    No Replicated UDT Columns
  Name: rep12, Id: 6553602 State: ACTIVE Flags: 0x800000 use 0
          lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab12
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
          sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 2064
    Column Name Verifcation: ON
    No Replicated UDT Columns
```

The following replicate information shows that the replicate belongs to a grid replicate set. `UTF8` indicates that code set conversion between replicates is enabled.

```
Name: grid_6553604_100_3, Id: 6553605 State: ACTIVE Flags: 0x900000 UTF8 GRID
        use 0 lastexec Wed Dec 31 18:00:00 1969

        Local Participant: tdb:nagaraju.t1
        Attributes: ROW scope, Enable RIS, all columns sent in updates
        Conflict resolution[Prim::Sec]: [ALWAYSAPPLY]
        Column Mapping: OFF
        Column Name Verifcation: ON
        No Replicated UDT Columns
```

This sample output from the onstat -g cat servers command shows that the server **g_bombay** and **g_delhi** are active; neither one is a hub or a leaf server, and both have ATS and RIS files that are generated in XML format.

```
GLOBAL-CATALOG CACHE STATISTICS

SERVERS
-------------------
  Current server  : Id 200, Nm g_bombay
  Last server slot: (0, 2)
  # free slots    : 0
  Broadcast map   : <[0005]>
  Leaf server map : <[0000]>
  Root server map : <[0006]>
  Adjacent server map: <[0004]>
    Id: 200, Nm: g_bombay, Or: 0x0002, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 00, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml

    Id: 100, Nm: g_delhi, Or: 0x0004, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 100, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml
```

Related reference

onstat -g cdr: Print ER statistics on page 550

Related information

Viewing grid information on page 131

onstat -g cdr: Print ER statistics

Prints the output for all of the Enterprise Replication statistics commands.

```
onstat  -gcdr
```

Usage

The output of the onstat -g cdr command is a combination of the following Enterprise Replication onstat command outputs:

- onstat -g cat
- onstat -g grp
- onstat -g que
- onstat -g rqm
- onstat -g nif all
- onstat -g rcv
- onstat -g dss
- onstat -g dtc
- onstat -g rep

Related reference

onstat -g cat: Print ER global catalog information on page 548

onstat -g grp: Print grouper statistics on page 556

onstat -g que: Print statistics for all ER queues on page 562

onstat -g rqm: Prints statistics for RQM queues on page 567

onstat -g nif: Print statistics about the network interface on page 561

onstat -g rcv: Print statistics about the receive manager on page 564

onstat -g dss: Print statistics for data sync threads on page 555

onstat -g dtc: Print statistics about delete table cleaner on page 555

onstat -g rep: Prints the schedule manager queue on page 566

onstat -g cdr config: Print ER settings

Prints the settings of Enterprise Replication configuration parameters and environment variables that can be set with the CDR_ENV configuration parameter.

This command has the following formats:

```
onstat -g cdr config
onstat -g cdr config long
onstat -g cdr config parameter_name
onstat -g cdr config parameter_name long
onstat -g cdr config CDR_ENV
onstat -g cdr config CDR_ENV long
onstat -g cdr config CDR_ENV variable_name
onstat -g cdr config CDR_ENV variable_name long
```

The long option prints additional information about settings that can be useful for IBM® Support.

The following table describes *parameter_name* and *variable_name*.

Modifier	Description
parameter_name	The name of an Enterprise Replication configuration parameter
variable_name	The name of an Enterprise Replication environment variable

If you use onstat -g cdr config without any options, the settings of all Enterprise Replication configuration parameters and environment variables are included in the output. If you specify the CDR_ENV configuration parameter without an environment variable name, all Enterprise Replication environment variables are included in the output.

Example

The following sample output of the onstat -g cdr config ENCRYPT_CDR command shows the setting of the ENCRYPT_CDR configuration parameter:

```
onstat -g cdr config ENCRYPT_CDR


ENCRYPT_CDR configuration setting:          0
```

The following sample output of the onstat -g cdr config CDR_ENV command shows the settings of all Enterprise Replication environment variables:

Example

```
onstat -g cdr config CDR_ENV

CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:           0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:            0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:             0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:        0
    CDRSITES_731:
        CDRSITES_731 configuration setting:             [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:             [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:             [None configured]
```

Example

The following sample output of the onstat -g cdr config command shows the settings of all Enterprise Replication configuration parameters and CDR_ENV environment variables:

```
onstat -g cdr config

CDR_DBSPACE:
    CDR_DBSPACE configuration setting:                rootdbs
CDR_DSLOCKWAIT:
    CDR_DSLOCKWAIT configuration setting:             5
CDR_EVALTHREADS:
    CDR_EVALTHREADS configuration setting:            1, 2
CDR_MAX_DYNAMIC_LOGS:
    CDR_MAX_DYNAMIC_LOGS configuration setting:       0
```

```
CDR_NIFCOMPRESS:
    CDR_NIFCOMPRESS configuration setting:            0
CDR_QDATA_SBSPACE:
    CDR_QDATA_SBSPACE configuration setting:          cdrsbsp
CDR_QHDR_DBSPACE:
    CDR_QHDR_DBSPACE configuration setting:           rootdbs
CDR_QUEUEMEM:
    CDR_QUEUEMEM configuration setting:               4096
CDR_SERIAL:
    CDR_SERIAL configuration setting:                 0, 0
CDR_SUPPRESS_ATSRISWARN:
    CDR_SUPPRESS_ATSRISWARN configuration setting:    [None suppressed]
ENCRYPT_CDR:
    ENCRYPT_CDR configuration setting:                0
ENCRYPT_CIPHERS:
    ENCRYPT_CIPHERS configuration setting:            [None configured]
ENCRYPT_MAC:
    ENCRYPT_MAC configuration setting:                [None configured]
ENCRYPT_MACFILE:
    ENCRYPT_MACFILE configuration setting:            [None configured]
ENCRYPT_SWITCH:
    ENCRYPT_SWITCH configuration setting:             0,0
CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:           0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:            0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:             0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:        0
    CDRSITES_731:
        CDRSITES_731 configuration setting:             [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:             [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:             [None configured]
```

Related information

Dynamically Modifying Configuration Parameters for a Replication Server on page 162

onstat -g ddr: Print status of ER log reader

Prints the status of the Enterprise Replication database log reader.

The **ddr**, or **ddr_snoopy**, is an internal component of Enterprise Replication that reads the log buffers and passes information to the grouper.

You can use the information from the **onstat -g ddr** command to monitor *replay position* in the log file and ensure replay position is never overwritten (which can cause loss of data). The replay position is the point from where, if a system failure occurs, Enterprise Replication starts re-reading the log information into the log update buffers. All the transactions generated before this position at all the target servers have been applied by Enterprise Replication or safely stored in stable queue space. As messages are acknowledged or stored in the stable queue, the replay position should advance. If you notice that replay position is not advancing, this can mean that the stable queue is full or a remote server is down.

The **onstat -g ddr** output shows you a snapshot of the replay position, the *snoopy position*, and the *current position*. The snoopy position identifies the position of the **ddr_snoopy** thread in the logical logs. The **ddr_snoopy** has read the log records up until this point. The current position is the position where the server has written its last logical log record.

If log reading is blocked, data might not be replicated until the problem is resolved. If the block is not resolved, the database server might overwrite the read (**ddr_snoopy**) position, which means that data will not be replicated. If this occurs, you must manually resynchronize the source and target databases.

To avoid these problems, follow these guidelines:

- Have 24 hours of online log space available.
- Keep the log file size consistent. Instead of having a single large log file, implement several smaller ones.
- Avoid switching logical logs more than once per hour.
- Keep some distance between LTXHWM (long-transaction high-watermark) and LTXEHWM (long-transaction, exclusive-access, high-watermark).

You can configure one or more actions to occur if the current position reaches the log needs position by setting the CDR_LOG_LAG_ACTION configuration parameter.

The following sample output from the **onstat ddr** command shows the replay position, snoopy position, and current position highlighted.

Example

```
DDR -- Running --

# Event  Snoopy   Snoopy   Replay   Replay   Current  Current
Buffers   ID      Position  ID      Position   ID     Position
2064     35       2ae050   34       121018   55       290000

Log Pages Snooped:
      From      From      From Staging     Tossed
     Cache      Disk      File             (LBC full)
         0         0       19704           0

CDR log records ignored   : 0
DDR log lag state   : On
Current DDR log lag action   : logstage
DDR log staging disk space usage  :0.26%
Maximum disk space allowed for log staging :1048576 KB
Maximum disk space ever used for log staging :2746.98 KB
Current staged log file count :21
Total dynamic log requests: 0

DDR events queue

Type   TX id    Partnum  Row id
```

Related reference

cdr view on page 486

onstat -g dss: Print statistics for data sync threads

Prints detailed statistical information about the activity of individual data sync threads.

The data sync thread applies the transaction on the target server. Statistics include the number of applied transactions and failures and when the last transaction from a source was applied.

The **onstat -g dss** command has the following formats:

```
onstat -g dss
onstat -g dss modifier
```

The following table describes the values for *modifier*.

Modifier	Action
UDR	Prints summary information about any UDR invocations by the data sync threads.
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the data sync threads. The `Procid` column lists the UDR procedure ID.

In the following example, only one data sync thread is currently processing the replicated data. It has applied a total of one replicated transaction and the transaction was applied at 2004/09/13 18:13:10. The Processed Time field shows the time when the last transaction was processed by this data sync thread.

```
-- Up 00:00:28 -- 28672 Kbytes
DS thread statistic
cmtTime    Tx       Tx        Tx       Last Tx
Name       < local  Committed Aborted  Processed  Processed Time
---------- -------  --------- -------  ---------  -----------------
CDRD_1     0        1         0        1          (1095117190) 2004/09/13
18:13:10
            Tables (0.0%):
            Databases: test
CDR_DSLOCKWAIT = 1
CDR_DSCLOSEINTERVAL = 60
```

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g dtc: Print statistics about delete table cleaner

Prints statistics about the delete table cleaner.

The delete table cleaner removes rows from the delete table when they are no longer needed.

The onstat -g dtc command is used primarily as a debugging tool and by IBM® Software Support.

In the following example, the thread name of the delete table cleaner is **CDRDTCleaner**. The total number of rows deleted is **1**. The last activity on this thread occurred at 2010/08/13 18:47:19. The delete table for replicate **rep1** was last cleaned at 2010/08/13 18:28:25.

```
-- Up 00:59:15 -- 28672 Kbytes
-- Delete Table Cleanup Status as of (1095119368) 2010/08/13 18:49:28
thread           = 49 <CDRDTCleaner>
      rows deleted     = 1
      lock timeouts    = 0
      cleanup interval = 300
      list size        = 3
      last activity    = (1095119239) 2010/08/13 18:47:19

Id      Database                         Last Cleanup Time
        Replicate            Server              Last Log Change
=========================================================
000001  test                             (1095118105) 2010/09/13
  18:28:25
        rep1                 g_bombay            (1095118105) 2010
        /08/13 18:28:25
        rep1                 g_delhi             (1095118105) 2010
        /08/13 18:28:25
000002  test                             <never cleaned>
```

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g grp: Print grouper statistics

Prints statistics about the grouper.

The grouper evaluates the log records, rebuilds the individual log records into the original transaction, packages the transaction, and queues the transaction for transmission.

Theonstat -g grp command is used primarily as a debugging tool and by IBM® Software Support.

The onstat -g grp command can take an optional modifier. The following table describes the values for the modifier.

Table 37. Modifiers for the onstat -g grp command

Modifier	Action
A	Prints all the information printed by the G, T, P, E, R, and S modifiers
E	Prints grouper evaluator statistics
Ex	Prints grouper evaluator statistics, expands user-defined routine (UDR) environments
G	Prints grouper general statistics

Table 37. Modifiers for the onstat -g grp command (continued)

Modifier	Action
L	Prints grouper global list
Lx	Prints grouper global list, expands open transactions
M	Prints grouper compression statistics
Mz	Clears grouper compression statistics
P	Prints grouper table partition statistics
pager	Prints grouper paging statistics
R	Prints grouper replicate statistics
S	Prints grouper serial list head (The serial list head is the first transaction in the list, that is, the next transaction that will be placed in the send queue.)
Sl	Prints grouper serial list (The serial list is the list of transactions, in chronological order.)
Sx	Prints grouper serial list, expands open transactions
T	Prints grouper transaction statistics
UDR	Prints summary information about any UDR invocations by the grouper threads
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the grouper threads The `Procid` column lists the UDR procedure ID.

The following sample shows output for the onstat -g grp command:

```
Grouper at 0xb014018:
Last Idle Time: (1095122236) 2010/09/13 19:37:16
RSAM interface ring buffer size: 528
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 48
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 5
Log update buffer memory in use: 0
Max log update buffer memory used at once: 320
Updates from Log: 16
Log update links allocated: 512
Blob links allocated: 0
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
Last Tx to Queuer began : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer ended : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer log ID, position: 12,23
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 2
Tx returned from Queuer: 2
Events sent to Queuer: 7
Events returned from Queuer: 7
Total rows sent to Queuer: 2
Open Tx array size: 1024
Table 'tab' at 0xae8ebb0 [ CDRShadow ]
Table 'tab12' at 0xae445e0 [ CDRShadow ]

Grouper Table Partitions:
  Slot  312...
    'tab' 1048888
  Slot  770...
    'tab12' 3145730
  Slot 1026...
    'tab12' 4194306
Repl links on global free list: 2
Evaluators: 3
  Evaluator at 0xb03d030 ID 0 [Idle:Idle] Protection:unused
    Eval iteration: 1264
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xb03d080
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d0d8 ID 1 [Idle:Idle] Protection:unused
    Eval iteration: 1265
    Updates evaluated: 2
    Repl links on local free list: 254
    UDR environment table at 0xb03d128
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d180 ID 2 [Idle:Idle] Protection:unused
    Eval iteration: 1266
    Updates evaluated: 4
    Repl links on local free list: 256
    UDR environment table at 0xb03d1d0
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
        Total Free Repl links 768

Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
```

```
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

The following example shows output for the onstat -g grp E command. The field **Evaluators: 4** indicates that there are four evaluation threads configured for the system.

```
Repl links on global free list: 0 Evaluators: 4
  Evaluator at 0xba71840 ID 0 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71890
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0
  Evaluator at 0xba718f0 ID 1 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71940
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0

  Evaluator at 0xba8c260 ID 2 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba8c2b0
        Number of environments:          0
        Table memory limit    :      16777
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xbaac2a0 ID 3 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xbaac2f0
        Number of environments:          0
        Table memory limit    :      16777
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
Total Free Repl links 1024
```

The following example shows output for the onstat -g grp G command.

```
Grouper at 0xb8ab020:
Last Idle Time: (1095115397) 2010/09/13 17:43:17
```

```
RSAM interface ring buffer size: 1040
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 64
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 1
Log update buffer memory in use: 0
Max log update buffer memory used at once: 64
Updates from Log: 1
Log update links allocated: 512
Blob links allocated: 0
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
```

The following example shows output for the onstat -g grp P command. In the following example, the grouper is evaluating rows for the **account**, **teller**, and **customer** tables.

```
Table 'teller' at 0xb851480 [ CDRShadow VarChars ]
Table 'account' at 0xb7faad8 [CDRShadow VarChars VarUDTs Floats
       Blobs]
Table 'customer' at 0xbbe67a8 [CDRShadow VarChars VarUDTs]
Grouper Table Partitions:
  Slot  387...
    'account' 1048707
  Slot  389...
    'teller' 1048709
  Slot  394...
    'customer' 1048714
```

The following example shows output for the onstat -g grp pager command. The sample output shows the grouper large transaction evaluation statistics.

```
Grouper Pager statistics:
Number of active big transactions: 0
Total number of big transactions processed: 0
Spool size of the biggest transaction processed: 0 Bytes
```

The following example shows output for the onstat -g grp R command. In this example, the grouper is configured to evaluate rows for replicates with IDs **6553601** and **6553602** (you can use the onstat -g cat repls command to obtain the replicate names). The **Ignore** attribute of replicate ID **6553602** shows that the grouper is currently not evaluating rows for this replicate. This can happen if the replicate state is not ACTIVE. You can obtain the replicate state using the onstat -g cat repls command.

```
Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

The following example shows output for the onstat -g grp T command. In this example, the grouper evaluated and queued 1 transaction to the send queue. The **Tx sent to Queuer** field shows the total number of transactions evaluated and queued to the send queue for propagating to all the replicate participants. The **Total rows sent to Queuer** field shows the total number of rows queued to the send queue for propagating to all the replicate participants.

```
Last Tx to Queuer began : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer ended : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer log ID, position: 5,3236032
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 1
Tx returned from Queuer: 0
Events sent to Queuer: 0
Events returned from Queuer: 0
Total rows sent to Queuer: 1
Open Tx array size: 1024
```

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g nif: Print statistics about the network interface

Prints statistics about the network interface.

```
onstat -g nif [ { all | sites | server_ID | sum } ]
```

The output shows which sites are connected and provides a summary of the number of bytes sent and received by each site. This can help you determine if a site is not sending or receiving bytes.

The onstat -g nif option is used primarily as a debugging tool and by IBM® Software Support.

The following table describes the options for onstat -g nif command:

Table 38. Options for the onstat -g nif command

Option	Action
all	Prints the sum and the sites.
sites	Prints the NIF site context blocks.
server_ID	Prints information about the replication server with that server ID.
sum	Prints the sum of the number of buffers sent and received for each site.

Example Output

The following example shows output for the onstat -g nif command. In this example, the local server is connected to the server group **g_bombay** and its CDR ID is **200**. The connection status is running. The connection between the two servers is running, but the replication state on the **g_bombay** server is suspended. The server group **g_bombay** internal NIF version is **9**. The local server has sent three messages to the server **g_bombay** and it has received two messages from **g_bombay**.

```
$ onstat -g nif

NIF anchor Block: af01610
       nifGState         RUN
          RetryTimeout   300

CDR connections:
 Id    Name       State    Version    Sent   Received
 -----------------------------------------------------
 200 g_bombay     RUN,SUSPEND    9       3          2
```

Output Description

NIF anchor Block

The address of the network storage block.

nifGState

The connection state.

RetryTimeout

The number of seconds before Enterprise Replication attempts to retry a dropped connection.

Id

The Enterprise Replication ID number for the server.

Name

The name of the server group.

State

The connection state between the local server and the listed server. If multiple states are shown the second state designates the replication state.

Version

The internal version number of the NIF component on the listed server.

Sent

The number of messages the local server has sent to the listed server.

Received

The number of messages received by the local server from the listed server.

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g que: Print statistics for all ER queues

Prints statistics that are common to all queues.

The queuer manages the logical aspects of the queue. The RQM (reliable queue manager) manages the physical queue.

The onstat -g que command is used primarily as a debugging tool and by IBM® Software Support.

In the following example, **Element high water mark** shows the maximum size of the transaction buffer header data (metadata) allowed in memory, shown in kilobytes. **Data high water mark** shows the maximum size of transactions for user data allowed in memory, shown in kilobytes.

```
CDR Queuer Statistics:
  Queuer state            : 2
  Local server            : 100
  Element high water mark : 131072
  Data high water mark    : 131072
  # of times txns split   : 0
  Total # of split txns   : 0
  allowed log delta       : 30
  maximum delta detected  : 4
  Control Key             : 0/00000007
  Synchronization Key     : 0/00000003
Replay Table:
  Replay Posn (Disk value): 12/00000018 (12/00000018)
  Replay save interval    : 10
  Replay updates          : 10
  Replay # saves          : 17
  Replay last save time   : (1095118157) 2010/09/13 18:29:17
Send Handles
  Server ID               : 200
  Send state,count        : 0,0
  RQM hdl for trg_send: Traverse handle (0xaf8e018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  RQM hdl for control_send:  Traverse handle (0xaf74018)
    for thread CDRACK_0 at Head_of_Q,  Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc6018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  Server ID               : 200
  Send state,count        : 0,0
  RQM hdl for trg_send: Traverse handle (0xac8b018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for control_send: Traverse handle (0xb1ce018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc5018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  Server ID               : 200
  Send state,count        : 0,0
  RQM hdl for trg_send: Traverse handle (0xaea71d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
  RQM hdl for ack_send: Traverse handle (0xae8c1d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
  RQM hdl for control_send: Traverse handle (0xae9e1d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
```

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g rcv: Print statistics about the receive manager

Prints statistics about the Enterprise Replication receive manager. The receive manager is a set of service routines between the receive queues and data sync. The onstat -g rcv command is used primarily as a debugging tool and by IBM® Software Support. If you suspect that acknowledgment messages are not being applied, you can run this command.

Syntax

```
onstat -grcv [{ serverid | full }]
```

The *serverID* option specifies a replication server.

The full option prints all statistics.

The following table describes the fields in the Receive Manager global section of the onstat -g rcv command output.

Table 39. Receive Manager section of the onstat -g rcv output

Field	Description
cdrRM_DSParallelPL	Shows the current level of Apply Parallelism, 0 (zero) being the highest
cdrRM_DSNumLockTimeout cdrRM_DSNumLockRB cdrRM_DSNumDeadLocks	Indicate the number of collisions between various apply threads
cdrRM_acksinList	Shows acknowledgments that are received but not yet processed

The Receive Parallelism Statistics section of the onstat -g rcv command output shows a summary of the data sync threads by source server.

Table 40. Receive Parallelism Statistics section of the onstat -g rcv output

Field	Description
Server	Source server ID
Concur	Number of transactions currently being applied in parallel
Tot.Txn.	Total number of transactions that are applied from this source server
Pending	Number of current transactions in the pending list for this source server
Active	Number of current transactions currently being applied from this source server
MaxPnd	Maximum number of transactions in the pending list queue
MaxAct	Maximum number of transactions in the active list queue
AvgPnd	Average depth of the pending list queue
AvgAct	Average depth of the active list queue
CommitRt	Commit rate of transaction from this source server, based on transactions per second

The Statistics by Source section of the onstat -g rcv command output shows the following information for each source server. For each replicate ID:

- The number of transactions that are applied from the source servers
- The number of inserts, deletes, and updates within the applied transactions
- The timestamp of the most recently applied transaction on the target server
- The timestamp of the commit on the source server for the most recently applied transaction

The following example shows output for the onstat -g rcv full command.

```
Receive Manager global  block 0D452018
    cdrRM_inst_ct:                      5
    cdrRM_State:              00000000
    cdrRM_numSleepers:        3
    cdrRM_DsCreated:          3
    cdrRM_MinDSThreads:       1
    cdrRM_MaxDSThreads:       4
    cdrRM_DSBlock             0
    cdrRM_DSParallelPL        0
    cdrRM_DSFailRate          0.000000
    cdrRM_DSNumRun:           35
    cdrRM_DSNumLockTimeout    0
    cdrRM_DSNumLockRB         0
    cdrRM_DSNumDeadLocks      0
    cdrRM_DSNumPCommits       0
    cdrRM_ACKwaiting          0
    cdrRM_totSleep:           77
    cdrRM_Sleeptime:          153
    cdrRM_Workload:           0
    cdrRM_optscale:           4
    cdrRM_MinFloatThreads:    2
    cdrRM_MaxFloatThreads:    7
    cdrRM_AckThreadCount:     2
    cdrRM_AckWaiters:         2
    cdrRM_AckCreateStamp:Wed Sep 08 11:47:49 2010
    cdrRM_DSCreateStamp: Wed Sep 08 14:16:35 2010
    cdrRM_acksInList:         0
    cdrRM_BlobErrorBufs:      0

 Receive Parallelism Statistics
 Server Concur Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
      1      8       1        0      0      1      1    1.00    1.00     0.06

 Tot Pending:0     Tot Active:0  Avg Pending:1.00    Avg Active:1.00
 Commit Rate:0.06

 Time Spent In RM Parallel Pipeline Levels
 Lev. TimeInSec  Pcnt.
    0        40 100.00%
    1         0   0.00%
    2         0   0.00%

 Statistics by Source

 Server 1
```

```
Repl      Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
65551       1       0       0       2 2012/10/29 09:52:23 2012/10/29 09:52:22

No Replicates Currently Being Throttled
```

If a replicate encounters a deadlock situation or otherwise reduces the degree of parallelism by which transactions are applied, the Statistics by Source section shows the replicate and the maximum number of concurrent transactions that are possible.

```
Statistics by Source

Server 1
Repl      Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
65551       1       0       0       2 2012/10/29 09:52:23 2012/10/29 09:52:22

Replicates Being Throttled
Repid      Max
           Concurrent
65551      3
```

If the replicate includes a **TimeSeries** column, a TimeSeries Statistics by Source section shows statistics about the time series elements that are applied on target servers:

```
TimeSeries Statistics by Source

Server 100
Repl        Txn   TSIns   TSDel   TSCmd Last Target Apply   Last Source Commit
65536       672     672       0       0 2012/08/27 15:04:33 2012/08/27 15:04:32
```

Related reference

onstat -g cdr: Print ER statistics on page 550

onstat -g rep: Prints the schedule manager queue

Prints events that are in the queue for the schedule manager.

The onstat -g rep command is used primarily as a debugging tool and by IBM® Software Support.

Theonstat -g rep command takes an optional replicate name to limit the output to those events originated by the specified replicate.

The following example shows sample output for the onstat -g rep command:

```
Schedule manager Cb: add7e18 State: 0x8100 <CDRINIT,CDRRUNNING>

Event        Thread             When
============================================
CDRDS          CDREvent       00:00:20
```

Related reference

cdr swap shadow on page 475

onstat -g cdr: Print ER statistics on page 550

onstat -g rqm: Prints statistics for RQM queues

Prints statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).

The RQM manages the insertion and removal of items to and from the various queues. The RQM also manages spooling of the in-memory portions of the queue to and from disk. The onstat -g rqm command displays the contents of the queue, size of the transactions in the queue, how much of the queue is in memory and on disk, the location of various handles to the queue, and the contents of the various progress tables. You can choose to print information for all queues or for just one queue by using one of the modifiers described below.

If a queue is empty, no information is printed for that queue.

The onstat -g rqm can take an optional modifier. The following table describes the values for the modifier.

Table 41. Values for the modifier to the onstat -g rqm command

Modifier	Action
ACKQ	Prints the ack send queue
CNTRLQ	Prints the control send queue
RECVQ	Prints the receive queue
SBSPACES	Prints detailed statistical information about the sbspaces configured for CDR_QDATA_SBSPACE.
SENDQ	Prints the send queue
SYNCQ	Prints the sync send queue
FULL	Prints full information about every in-memory transaction for every queue
BRIEF	Prints a brief summary of the number of transactions in each of the queues and the replication servers for which the data is queued Use this modifier to quickly identify sites where a problem exists. If large amounts of data are queued for a single server, then that server is probably down or off the network.
VERBOSE	Prints all the buffer headers in memory

When you specify a modifier to select a specific queue, the command prints all the statistics for that queue and information about the first and last in-memory transactions for that queue. When you select the SBSPACES modifier, the command prints information about the sbspaces being used for replication, including how full those sbspaces are.

The other modifiers of the onstat -g rqm command are used primarily as a debugging tool and by Technical Support.

The output for the SENDQ modifier contains the following sections:

- The current statistics section (`Transaction spool name` through `Pending Txn Data`): Contains information about the current contents of the queue, such as how many bytes are contained in the queue, how many transactions are in the queue, how many transactions are currently in memory, how many have been spooled to disk, how many exist only on disk, and so on. The `Insert Stamp` field value is used to maintain the order of the transactions within the queue. The `Size of Data in queue` field shows the size of the queue when combining the in-memory transactions with the spool-only transactions. The `Pending Txn Buffers` field contains information about transactions that are in the process of being queued into the send queue.
- The historical statistics section (`Max Real memory data used` through `Total Txn Lookups`): contains a summary of what has been placed in the queue in the past. The `Max Real memory data used` field contains the largest in memory size of the queue. The `Total Txn Recovered` field shows the transactions that existed only in the spool when the server was started. The `Total Txns deleted` field shows the number of transactions that have been removed from the queue. The `Total Txns duplicated` field contains the number of times attempted to queue a transaction that had already been processed. The `Total Txn Lookups` field is a counter of the number of times that an Enterprise Replication thread attempted to read a transaction.
- The `Progress Table` section: contains information on what is currently queued, to which server it is queued for, and what has been acknowledged from each of the participants of the replicate. The first part of the progress table section is a summary. Below the summary section is a list of the servers and group entries that contain what is currently queued for each server, what has been sent to the remote server, and what has been acknowledged from the remote server. The contents of the `ACKed` and `Sent` columns contains the key of the last transaction that was acknowledged from the remote server or sent to that server. The key is a multi-part number consisting of *source_node/unique_log_id/logpos/incremental number*. The transaction section contains the first and last transaction in the queue that are currently in memory. The `NeedAck` field shows from which server the transaction is waiting for an acknowledgment. You can use this bitmap mask with the output from the onstat -g cat command to determine the name of the server which server Enterprise Replication is waiting on for an acknowledgment.
- The `Transverse handle` section: contains the position within the queue that any thread is currently processing. Each thread that attempts to read a transaction from the queue, or to place a transaction into the queue must first allocate a handle. This handle is used to maintain the positioning within the queue.

The following example shows output for the onstat -g rqm SENDQ command.

Example

```
> onstat  -g rqm SENDQ



CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0xb956020) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 9/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:             0
 Log Events in queue:       0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 385830 (4194304) Bytes
 Max Real memory hdrs used  23324 (4194304) Bytes
 Total data queued:         531416 Bytes
 Total Txns queued:         9
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        9
 Total Txns duplicated:     0
 Total Txn Lookups:         54

 Progress Table:
        Progress Table is Stable
                On-disk table name............:         spttrg_send
                Flush interval (time).........:         30
                Time of last flush............:         1207866706
                Flush interval (serial number):         1000
                Serial number of last flush...:         1
                Current serial number.........:         5

Server    Group Bytes Queued      Acked                     Sent
------------------------------------------------------------------------------
    20  0xa0002      12 efffffff/efffffff/efffffff/efffffff -  a/e/1510a1/0
    20  0xa0003                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    30  0xa0004                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    20  0xa0004                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    20  0xa0001                  0 a/d/6e81f8/0             -  a/d/6e81f8/0



First Txn (0x0D60C018) Key:  1/9/0x000d4bb0/0x00000000
 Txn Stamp: 1/0, Reference Count: 0.
 Txn Flags: Notify
 Txn Commit Time: (1094670993) 2004/09/08 14:16:33
 Txn Size in Queue: 5908
 First Buf's (0x0D31C9E8) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 NeedAck: Waiting for Acks from <[0004]>
 No open handles on txn.

 Last Txn (0x0D93A098) Key:  1/9/0x00138ad8/0x00000000
 Txn Stamp: 35/0, Reference Count: 0.
 Txn Flags: Notify
 Txn Commit Time: (1094671237) 2004/09/08 14:20:37
 Txn Size in Queue: 6298
 First Buf's (0x0D92FFA0) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 NeedAck: Waiting for Acks from <[0004]>
 Traverse handle (0xbca1a18) for thread CDRGeval0 at Head_of_Q,  Flags: None
 Traverse handle (0xb867020) for thread CDRACK_1 at Head_of_Q,  Flags: None
 Traverse handle (0xbcbd020) for thread CDRGeval1 at Head_of_Q,  Flags: None
 Traverse handle (0xbd08020) for thread CDRGeval3 at Head_of_Q,  Flags: None
 Traverse handle (0xbe511c8) for thread CDRGeval2 at Head_of_Q,  Flags: None
 Traverse handle (0xbe58158) for thread CDRACK_0 at Head_of_Q,  Flags: None
```

The following output is an example of the onstat -g rqm SBSPACES command.

Example

```
onstat -g rqm sbspaces

Blocked:DDR



RQM Space Statistics for CDR_QDATA_SBSPACE:
-------------------------------------------
name/addr      number    used      free   total    %full   pathname
0x46581c58     5         311       1      312      100     /tmp/amsterdam_sbsp_base
amsterdam_sbsp_base5     311       1      312      100

0x46e54528     6         295       17     312      95      /tmp/amsterdam_sbsp_2
amsterdam_sbsp_26        295       17     312      95

0x46e54cf8     7         310       2      312      99      /tmp/amsterdam_sbsp_3
amsterdam_sbsp_37        310       2      312      99

0x47bceca8     8         312       0      312      100     /tmp/amsterdam_sbsp_4
amsterdam_sbsp_48        312       0      312      100
```

In this example, the sbspaces are all either full or nearly full.

Related reference

onstat -g cdr: Print ER statistics on page 550

Related information

Monitoring Disk Usage for Send and Receive Queue Spool on page 215

onstat -g sync: Print statistics about synchronization

Prints statistics about the active synchronization process.

The following example shows output for the onstat -g sync command.

```
Prim   Sync   St.  Shadow Flag Stat Block  EndBlk
Repl   Source      Repl             Num    Num

655361 20     0    1310729 2    0    592    600
```

Output Description

Prim Repl

Replicate number of the replicate being synchronized

Sync Source

Source server of the sync

St

Sync replicate state

Shadow Repl

The shadow replicate used to perform the sync

Flag

Internal flags:

- 0x02 = external sync
- 0x04 = shutdown request has been issued
- 0x08 = abort has occurred
- 0x010 = a replicate stop has been requested
- 0X020 = shadow or primary replicate has been deleted

Stat

Resync job state

Block num

Last block applied on targets (on source always 0)

EndBlock Num

Last block in resync process. Marks the end of the sync scan on the target. A value of' -2 indicates that the scan is still in progress, and the highest block number is not yet known.

Additional fields for forwarded rows:

ServID

Server where forwarded row originated

fwdLog ID

Originator's log ID of the forwarded row

fwdLog POS

Originator's log position of the forwarded row

endLog ID

Operation switches back to normal at this point

endLog POS

Operation switches back to normal at this log position

complete flag

Set to 1 after normal processing resumes for the originating source

syscdr Tables

These tables in the **syscdr** database contain progress information about consistency checking and synchronization operations.

The replcheck_stat Table

The **replcheck_stat** table contains the progress information for consistency check and synchronization operations that specified a progress report task name.

Column	**Type**	**Purpose**
replcheck_id	serial	Unique key for the task and replicate combination.
replcheck_name	varchar(32)	The task name.
replcheck_replname	varchar(128)	The replicate name.
replcheck_type	char(1)	The task type: • C = consistency check • S = synchronization
replcheck_numrows	integer	The total number of rows in the table.
replcheck_rows_processed	integer	The number of rows that were processed to correct inconsistent rows.
replcheck_status	char(1)	The status of this task: • A = Aborted • D = Defined • R = Running • C = Completed • F = Completed, but inconsistent • W = Pending complete
replcheck_start_time	datetime year to second	The time that the sync or check task for the replicate started running.
replcheck_end_time	datetime year to second	The time that sync or check task for the replicate completed.

Related reference

cdr stats check on page 457

cdr stats sync on page 461

The replcheck_stat_node Table

The **replcheck_stat_node** table contains the progress information for the consistency check and synchronization operations with progress report task names on a particular replication server.

Column	Type	Purpose
replnode_replcheck_id	integer	Server group ID (CDR ID).
replcheck_node_id	integer	Unique key for the task, replicate, and server combination.
replcheck_order	integer	A number to provide consistent ordering for display purposes.
replcheck_node_name	varchar(128)	The name of the replication server.
replnode_table_owner	varchar(128)	The owner of table being synchronized or checked.
replnode_table_name	varchar(128)	The name of the table being synchronized or checked.
replnode_row_count	integer	The number of rows in the participant.
replnode_processed_rows	integer	The number of rows processed to correct inconsistent rows.
replnode_missing_rows	integer	The number of rows on the reference server that do not exist on the target server.
replnode_extra_rows	integer	The number of rows on the target server that do not exist on the reference server.
replnode_mismatched_rows	integer	The number of rows on the target server that are not consistent with the corresponding rows on the reference server.
replnode_extra_child_rows	integer	The number of child rows that required processing on the target nodes.

Related reference

cdr stats check on page 457

cdr stats sync on page 461

SMI Tables for Enterprise Replication Reference

The system-monitoring interface (SMI) tables in the **sysmaster** database provide information about the state of the database server. Enterprise Replication uses the following SMI tables.

Related information

Monitor and troubleshooting Enterprise Replication on page 197

The syscdr_ats Table

The **syscdr_ats** table contains the first ten lines of the transaction header for each ATS file.

Column	Type	Description
ats_ris	integer	Pseudo row ID.
ats_file	char(128)	ATS file name.
ats_sourceid	integer	CDRID of source server.
ats_source	char(128)	Source server name.
ats_committime	char(20)	Time when the transaction was committed on the source server.
ats_targetid	integer	CDRID of the target server.
ats_target	char(128)	Target server name.
ats_receivetime	char(20)	Time when the transaction was received on the target server .
ats_risfile	char(128)	Corresponding RIS file name.
ats_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ats_line5	char(200)	The fifth line of the transaction header information.
ats_line6	char(200)	The sixth line of the transaction header information.
ats_line7	char(200)	The seventh line of the transaction header information.
ats_line8	char(200)	The eighth line of the transaction header information.

Column	Type	Description
ats_line9	char(200)	The ninth line of the transaction header information.
ats_line10	char(200)	The tenth line of the transaction header information.

The syscdr_atsdir Table

The **syscdr_atsdir** table contains information about the contents of the ATS directory.

Column	Type	Description
atsd_rid	integer	Pseudo row ID
atsd_file	char(128)	ATS file name
atsd_mode	integer	File mode
atsd_size	integer	File size in bytes
atsd_atime	datetime	Last access time
atsd_mtime	datetime	Last modified time
atsd_ctime	datetime	Create time

The syscdr_ddr Table

The **syscdr_ddr** table contains information about the status of log capture and the proximity or status of transaction blocking (DDRBLOCK) or transaction spooling.

Column	Type	Description
ddr_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
ddr_snoopy_loguniq	integer	The current log ID at which transactions are being captured for replication
ddr_snoopy_logpos	integer	The current log position at which transactions are being captured for replication
ddr_replay_loguniq	integer	The current log ID at which transactions have been applied
ddr_replay_logpos	integer	The current log position at which transactions have been applied. This is the position from which the log would need to be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.
ddr_curr_loguniq	integer	The current log ID

Column	Type	Description
ddr_curr_logpos	integer	The current log position
ddr_logsnoop_cached	integer	The number of log pages that log capture read from its cache
ddr_logsnoop_disk	integer	The number of times that log capture had to read log pages from disk
ddr_log_tossed	integer	The number of log pages that could not be stored in the cache because the log capture buffer cache was full
ddr_logs_ignored	integer	The number of log records that were ignored because they were extensible log records unknown to Enterprise Replication
ddr_dlog_requests	integer	The number of times that a dynamic log was requested to be created to prevent DDRBLOCK state
ddr_total_logspace	integer	The total number of log pages in the replication system
ddr_logspace2wrap	integer	The number of log spaces until log capture runs into a log wrap
ddr_logpage2block	integer	The number of log pages until log capture runs into a DDRBLOCK state
ddr_logneeds	integer	The number of log pages necessary to prevent a log wrap to avoid a DDRBLOCK state
ddr_logcatchup	integer	The number of log pages necessary to process before going out of a DDRBLOCK state
ddr_loglag_state	char(10)	The state of DDR log lag: on or off
ddr_cur_loglag_act	char(24)	The action being taken to prevent log wrapping
ddr_logstage_diskusage	float	The amount of used log staging disk space as a percentage of the total space
ddr_logstage_hwm4disk	integer	The maximum allowable disk space for log staging in KB
ddr_logstage_maxused	float	The maximum disk space ever used for log staging in KB
ddr_logstage_lfile_cnt	integer	The number of staged log files

The syscdr_nif Table

The **syscdr_nif** table contains information about network connections and the flow of data between Enterprise Replication servers.

Column	Type	Description
nif_connid	integer	The CDRID of the peer node
nif_connname	char(24)	The name (group name) of the peer node
nif_state	char(24)	The status of the Enterprise Replication network:

Column	Type	Description
		• Admin Close = Enterprise Replication was stopped by user by issuing the **cdr stop** command • Connected = The connection is active • Connecting = The connection is being established • Disconnected = The connection was explicitly disconnected • Local server = The connection is to the local server. • Logic Error = The connection disconnected due to an error during message transmission • Never Connected = The servers have never had an active connection • Start Error = The connection disconnected due to an error while starting a thread to receive remote messages • Timeout = The connection attempt has timed out, but will be reattempted
nif_connstate	char(24)	The connection state: • ABORT = The connection is being aborted. • BLOCK = The connection has been blocked from transmitting data by the other server. • INIT = The connection is being initialized. • INTR = The connection has been interrupted. • RUN = The connection is active. • SHUT = The connection is shutting down in an orderly way. • SLEEP = The connection is waiting to receive data. • STOP = The connection is stopped. • SUSPEND = The connection has been suspended. • TIMEOUT = The connection has timed out.
nif_version	integer	The network protocol of this connection used to convert the message formats between dissimilar releases of the server, for example, HCL Informix® 7 and HCL Informix® 9
nif_msgsent	integer	Number of messages sent to the peer server
nif_bytessent	integer	Number of bytes sent to the peer server
nif_msgrcv	integer	Number of messages received from the peer server
nif_bytesrcv	integer	Number of bytes received from the peer server
nif_compress	integer	Compression level for communications

Column	Type	Description
		• -1 = no compression • 0 = compress only if the target server expects compression • 1 - 9 = increasing levels of compression
nif_sentblockcnt	integer	Number of times a flow block request was sent to the peer server to delay sending any further replicated transactions for a short time because the receive queue on the target server is full
nif_rcvblockcnt	integer	Number of times a flow block request was received from the peer server
nif_trgsend_stamp1	integer	Stamp 1 of the last transaction sent to the peer server
nif_trgsend_stamp2	integer	Stamp 2 of the last transaction sent to the peer server
nif_acksend_stamp1	integer	Stamp 1 of the last acknowledgment sent to the peer server
nif_acksend_stamp2	integer	Stamp 2 of last acknowledgment sent to the peer server
nif_ctrlsend_stamp1	integer	Stamp 1 of the last control message sent to the peer server
nif_ctrlsend_stamp2	integer	Stamp 2 of the last control message sent to the peer server
nif_syncsend_stamp1	integer	Stamp 1 of the last sync message sent to the peer server
nif_syncsend_stamp2	integer	Stamp 2 of the last sync message sent to the peer server
nif_starttime	datetime	Time that the connection was established
nif_lastsend	datetime	Time of the last message sent to the peer server

The syscdr_rcv Table

The **syscdr_rcv** table contains information about transactions being applied on target servers and acknowledgments being sent from target servers.

Column	Type	Description
rm_state	char(100)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication
rm_num_sleepers	integer	Number of data sync threads currently suspended
rm__num_dsthreads	integer	The current number of data sync threads
rm_min_dsthreads	integer	Minimum number of data sync threads
rm_max_dsthreads	integer	Maximum number of data sync threads

Column	Type	Description
rm_ds_block	integer	If 1, the data sync is currently blocked to try to avoid causing a DDRBLOCK state
rm_ds_parallel	integer	The degree to which transactions are applied in parallel (0 through 3, inclusive): • 0 = the highest degree of parallelism • 3 = serial apply (no parallelism)
rm_ds_failrate	float	A computed weighted ratio that is used to determine when to change the degree of apply parallelism based on the rate of transactions that could not be applied
rm_ds_numrun	integer	Number of transactions run
rm_ds_lockout	integer	Number of lock timeouts encountered
rm_ds_lockrb	integer	Number of forced rollbacks due to having to switch to serial apply
rm_ds_num_deadlocks	integer	Number of deadlocks encountered
rm_ds_num_pcommits	integer	Number of out-of-order commits that have occurred
rm_ack_waiting	integer	Number of acknowledgments that are waiting for a log flush to return to the source server
rm_tosleep	integer	Total times that the data sync threads have become suspended
rm_sleeptime	integer	Total time that the data sync threads have been suspended
rm_workload	integer	The current workload
rm_optscale	integer	Factor determining how many data sync threads will be allowed per CPU VP
rm_min_fthreads	integer	Minimum acknowledgment threads
rm_max_fthreads	integer	Maximum acknowledgment threads
rm_ack_start	char(64)	Time when the acknowledgment threads started
rm_ds_start	char(64)	Time when the data sync threads started
rm_pending_acks	integer	Number of acknowledgments on the source that have not yet been processed
rm_blob_error_bufs	integer	Number of smart large objects that could not be successfully applied

The syscdr_ris Table

The **syscdr_ris** table contains the first ten lines of the transaction header for each RIS file.

Column	Type	Description
ris_rid	integer	Pseudo row ID.
ris_file	char(128)	RIS file name.
ris_sourceid	integer	CDRID of source server.
ris_source	char(128)	Source server name.
ris_committime	char(20)	Time when the transaction was committed on the source server.
ris_targetid	char(128)	CDRID of the target server.
ris_target	integer	Target server name.
ris_receivetime	char(20)	Time when the transaction was received on the target server.
ris_atsfile	char(128)	Corresponding ATS file.
ris_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ris_line5	char(200)	The fifth line of the transaction header information.
ris_line6	char(200)	The sixth line of the transaction header information.
ris_line7	char(200)	The seventh line of the transaction header information.
ris_line8	char(200)	The eighth line of the transaction header information.
ris_line9	char(200)	The ninth line of the transaction header information.
ris_line10	char(200)	The tenth line of the transaction header information.

The syscdr_risdir Table

The **syscdr_risdir** table contains information about the contents of the RIS directory.

Column	Type	Description
risd_rid	integer	Pseudo row ID
risd_file	char(128)	RIS file name
risd_mode	integer	File mode
risd_size	integer	File size in bytes
risd_atime	datetime	Last access time
risd_mtime	datetime	Last modified time
risd_ctime	datetime	Create time

The syscdr_rqm Table

The **syscdr_rqm** table contains statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).

The RQM manages the insertion and removal of items to and from the various queues. The RQM also manages spooling of the in-memory portions of the queue to and from disk.

Column	Type	Description
rqm_idx	integer	Index number
rqm_name	char(128)	Queue name
rqm_flags	integer	Flags
rqm_txn	integer	Transactions in queue
rqm_event	integer	Events in queue
rqm_txn_in_memory	integer	Transaction in memory
rqm_txn_in_spool_only	integer	Spool-only transactions
rqm_txn_spooled	integer	Spooled transactions
rqm_unspooled_bytes	int8	Unspooled bytes
rqm_data_in_queue	int8	Data in queue
rqm_inuse_mem	int8	Real memory in use
rqm_pending_buffer	integer	Pending buffers
rqm_pending_data	int8	Pending buffers
rqm_maxmemdata	int8	Maximum memory in use by data
rqm_maxmemhdr	int8	Maximum memory in use by headers
rqm_totqueued	int8	Total data queued
rqm_tottxn	integer	Total transactions queued
rqm_totspooled	integer	Total transactions spooled
rqm_totrestored	integer	Total transactions stored
rqm_totrecovered	integer	Total transactions recovered
rqm_totspoolread	integer	Total rows read from spool
rqm_totdeleted	integer	Total transactions deleted
rqm_totduplicated	integer	Total transactions duplicates
rqm_totlookup	integer	Total transaction lookups

The syscdr_rqmhandle Table

The **syscdr_rqmhandle** table contains information about which transaction is being processed in each queue. The handle marks the position of the thread in the queue.

Column	Type	Description
rqmh_qidx	integer	The queue associated with this handle
rqmh_thread	char(18)	Thread owning the handle
rqmh_stamp1	integer	Stamp 1 of the last transaction this handle accessed
rqmh_stamp2	integer	Stamp 2 of the last transaction this handle accessed
rqmh_servid	integer	Part 1 of the transaction key
rqmh_logid	integer	Part 2 of the transaction key
rqmh_logpos	integer	Part 3 of the transaction key
rqmh_seq	integer	Part 4 of the transaction key

The syscdr_rqmstamp Table

The **syscdr_rqmstamp** table contains information about which transaction is being added to each queue.

Column	Type	Description
rqms_qidx	integer	Queue index corresponding to the queues: • `0` = Transaction Send Queue • `1` = Acknowledgment Send Queue • `2` = Control Send Queue • `3` = CDR Metadata Sync Send Queue • `4` = Transaction Receive Queue
rqms_stamp1	integer	Stamp 1 of the next transaction being put into the queue
rqms_stamp2	integer	Stamp 2 of the next transaction being put into the queue
rqms_cstamp1	integer	Communal stamp 1 used to identify the next transaction read from the receive queue
rqms_cstamp2	integer	Communal stamp 2 used to identify the next transaction read from the receive queue

The syscdr_state Table

The **syscdr_state** table contains status on Enterprise Replication, data capture, data apply, and the network between the servers.

Column	Type	Description
er_state	char(24)	The status of Enterprise Replication: • Abort = Enterprise Replication is aborting on this server. • Active = Enterprise Replication is running normally. • Down = Enterprise Replication is stopped on this server. • Dropped = The attempt to drop the **syscdr** database failed. • Init Failed = The initial start-up of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server. • Initializing = Enterprise Replication is being defined. • Initial Startup = Enterprise Replication is starting for the first time on this server. • Shutting Down = Enterprise Replication is shutting down on this server. • Startup Blocked = Enterprise Replication cannot start because the server was started with the **oninit -D** command. • Synchronizing Catalogs = The server is receiving a copy of the **syscdr** database. • Uninitialized = The server does not have Enterprise Replication defined on it.
er_capture_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
er_network_state	char(64)	The status of the Enterprise Replication network: • Running = Communication is running normally • Down = Communication is not running • Uninitialized = The server is not a source server for replication
er_apply_state	char(24)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication

The syscdrack_buf Table

The **syscdrack_buf** table contains information about the buffers that form the acknowledgment queue.

When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue.

For information on the columns of the **syscdrack_buf** table, refer to Columns of the Buffer Tables on page 596.

The syscdrack_txn Table

The **syscdrack_txn** table contains information about the acknowledgment queue.

When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue. The acknowledgment queue is an in-memory only queue. That is, it is a volatile queue that is lost if the database server is stopped.

For information on the columns of the **syscdrack_txn** table, refer to Columns of the Transaction Tables on page 595.

The syscdrctrl_buf Table

The **syscdrctrl_buf** table contains buffers that provide information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_buf** table, refer to Columns of the Buffer Tables on page 596.

The syscdrctrl_txn Table

The **syscdrctrl_txn** table contains information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_txn** table, refer to Columns of the Transaction Tables on page 595.

The syscdrerror Table

The **syscdrerror** table contains information about errors that Enterprise Replication has encountered.

Column	Type	Description
errornum	integer	Error number
errorserv	char(128)	Database server name where error occurred
errorseqnum	integer	Sequence number that can be used to prune single-error table
errortime	datetime year to second	Time error occurred
sendserv	char(128)	Database server name, if applicable, that initiated error behavior
reviewed	char(1)	• `Y` if reviewed and set by DBA • `N` if not reviewed

Column	Type	Description
errorstmnt	text	Error description

The syscdrlatency Table

The **syscdrlatency** table contains statistics about Enterprise Replication latency (the time it takes to replicate transactions).

Column	Type	Description
source	integer	Source of transaction (**cdrid**)
replid	integer	Replicate ID
txncnt	integer	The number of transactions on this source replicate
inserts	integer	Number of INSERT statements
deletes	integer	Number of DELETE statements
updates	integer	Number of UPDATE statements
last_tgt_apply	integer	The time of the last transaction to be applied to the target (**cdrtime**)
last_src_apply	integer	The time of the last transaction to be applied on the source (**cdrtime**)

The syscdrpart Table

The **syscdrpart** table contains participant information.

Column	Type	Description
replname	lvarchar	Replicate name
servername	char(128)	Database server name
partstate	char(50)	Participant state: ACTIVE, INACTIVE
partmode	char(1)	• `P` = primary database server (read/write) • `R` = target database server (receive only)
dbsname	lvarchar	Database name
owner	lvarchar	Owner name
tabname	lvarchar	Table name
pendingsync	integer	• `0` = the Pending Sync attribute is not set • `1` = the Pending Sync attribute is set, indicating that the participant is waiting to be synchronized after the replication server was enabled

The syscdrprog Table

The **syscdrprog** table lists the contents of the Enterprise Replication progress tables.

The progress tables keep track of what data has been sent to which servers and which servers have acknowledged receipt of what data. Enterprise Replication uses the transaction keys and stamps to keep track of this information.

The progress table is two dimensional. For each server to which Enterprise Replication sends data, the progress tables keep progress information on a per-replicate basis.

Column	Type	Description
dest_id	integer	Server ID of the destination server
repl_id	integer	The ID that Enterprise Replication uses to identify the replicate for which this information is valid
source_id	integer	Server ID of the server from which the data originated
key_acked_srv	integer	Last key for this replicate that was acknowledged by this destination
key_acked_lgid	integer	Logical log ID
key_acked_lgpos	integer	Logical log position
key_acked_seq	integer	Logical log sequence
tx_stamp_1	integer	Together with tx_stamp2, forms the stamp of the last transaction acknowledged for this replicate by this destination
tx_stamp_2	integer	Together with tx_stamp1, forms the stamp of the last transaction acknowledged for this replicate by this destination

The syscdrq Table

The **syscdrq** table contains information about Enterprise Replication queues.

Column	Type	Description
srvid	integer	The identifier number of the database server
repid	integer	The identifier number of the replicate
srcid	integer	The server ID of the source database server In cases where a particular server is forwarding data to another server, *srvid* is the target and *srcid* is the source that originated the transaction.
srvname	char(128)	The name of the database server
replname	char(128)	Replicate name
srcname	char(128)	The name of the source database server

Column	Type	Description
bytesqueued	integer	Number of bytes queued

The syscdrqueued Table

The **syscdrqueued** table contains data-queued information.

Column	Type	Description
servername	char(128)	Sending to database server name
name	char(128)	Replicate name
bytesqueued	decimal(32,0)	Number of bytes queued for the server *servername*

The syscdrrecv_buf Table

The **syscdrrecv_buf** table contains buffers that provide information about the data-receive queue.

When a replication server receives replicated data from a source database server, it puts this data on the receive queue for processing. On the target side, Enterprise Replication picks up transactions from this queue and applies them on the target.

For information on the columns of the **syscdrrecv_buf** table, refer to Columns of the Buffer Tables on page 596.

The syscdrrecv_stats Table

The **syscdrrecv_stats** table contains statistics about the receive manager. The receive manager is a set of service routines between the receive queues and data sync.

Column	Type	Description
source	integer	The source server (**cdrid**)
txncnt	integer	Number of transactions from this source
pending	integer	The transaction currently pending on this source
active	integer	The transaction currently active on this source
maxpending	integer	Maximum pending transactions on this source
maxactive	integer	Maximum active transactions on this source
avg_pending	float	Average pending transactions on this source
avg_active	float	Average active transactions on this source
cmtrate	float	Average commit rate from this source

The syscdrrecv_txn Table

The **syscdrrecv_txn** table contains information about the data receive queue. The receive queue resides in memory.

When a replication server receives replicated data from a source database server, it puts the data in the receive queue and then applies the transactions on the target.

For information on the columns of the **syscdrrecv_txn** table, refer to Columns of the Transaction Tables on page 595.

The syscdrrepl Table

The **syscdrrepl** table contains replicate information.

Column	Type	Description
replname	lvarchar	Replicate name.
replstate	char(50)	Replicate state. For possible values, refer to cdr list server on page 402.
freqtype	char(1)	Type of replication frequency: • `C` = continuous • `I` = interval • `T` = time based • `M` = day of month • `W` = day of week
freqmin	smallint	The time for replication by minute: • Minutes after the hour that replication should occur. • Null if continuous.
freqhour	smallint	The time for replication by hour: • Hour that replication should occur. • Null if continuous.
freqday	smallint	Day of week or month replication should occur.
scope	char(1)	Replication scope: • `T` = transaction • `R` = row-by-row
invokerowspool	char(1)	Whether Row Information Spooling is enabled:

Column	Type	Description
		• `Y` = row spooling is enabled. • `N` = row spooling is disabled.
invoke transpool	char(1)	Whether Aborted Transaction Spooling is enabled: • `Y` = transaction spooling is enabled. • `N` = transaction spooling is disabled.
primresolution	char(1)	Type of primary conflict resolution: • `A` = always apply • `D` = delete wins • `I` = ignore • `T` = timestamp. • `S` = SPL routine
secresolution	char(1)	Type of secondary conflict resolution: • `S` = SPL routine • Null = not configured
storedprocname	lvarchar	SPL routine: • Name of SPL routine for secondary conflict resolution. • Null if not defined.
floattype	char(1)	Type of floating point number conversion: • `C`= converts floating point numbers to canonical format. • `I`= converts floating point numbers to IEEE format. • `N` = does not convert floating point numbers (sends in native format).
istriggerfire	char(1)	Whether triggers are enabled: • `Y` = triggers are enabled. • `N` = triggers are disabled.
isfullrow	char(1)	Whether to replicate full rows or only the changed columns:

Column	Type	Description
		• Y = sends the full row and enables upserts. • N = sends only changed columns and disables upserts.
isgrid	char(1)	Whether the replicate belongs to a grid replicate set: • Y = the replicate belongs to a grid replicate set. • N = the replicate does not belong to a grid replicate set.

Related information

Viewing grid information on page 131

The syscdrreplset Table

The **syscdrreplset** table contains replicate set information.

Column	Type	Description
replname	lvarchar	Replicate name
replsetname	lvarchar	Replicate set name
replsetattr	integer	Replicate set attributes: • 0x00200000U = The replicate set was created with a template. • 0x00000080U = The replicate set is exclusive.

The syscdrs Table

The **syscdrs** table contains information about database servers in an Enterprise Replication domain.

Column	Type	Description
servid	integer	Server identifier.
servname	char(128)	Database server name.
cnnstate	char(1)	Status of connection to this database server: • C = Connected • D = Connection disconnected (will be retried) • E = Error during connection • F = Connection failed

Column	Type	Description
		- K = In the process of connecting - L = The connection is to the local server - R = Disconnected but will attempt to reconnect - T = Idle time-out caused connection to terminate - X = Connection closed by user command and unavailable until reset by user
cnnstatechg	integer	Time that connection state was last changed.
servstate	char(1)	Status of database server: - A = Active. The server is active and replicating data. - D = Deleted. The server has been deleted; it is not capturing or delivering data and the queues are being drained. - S = Suspended. Delivery of replication data to the server is suspended. - Q = Quiescent. The server is in the process of being defined. - U = Disabled. Only delete shadow tables are populated in this state.
ishub	char(1)	Whether the server is a hub server that forwards information to another replication server: - Y = Server is a hub - N = Server is not a hub
isleaf	char(1)	Whether the server is a leaf or a nonleaf server: - Y = Server is a leaf server - N = Server is not a leaf server
rootserverid	integer	The identifier of the root server.
forwardnodeid	integer	The identifier of the parent server.
timeout	integer	The number of minutes of idle time between replication servers before the connection is timed out.

Although not directly connected, a nonroot server is similar to a root server except it forwards all replicated messages through its parent (root) server. All nonroot servers are known to all root servers and other nonroot servers. A nonroot server can be a terminal point in a tree or it can be the parent for another nonroot server or a leaf server. Nonroot and root servers are aware of all replication servers in the replication environment, including all the leaf servers.

A leaf server is a nonroot server that has a partial catalog. A leaf server has knowledge only of itself and its parent server. It does not contain information about replicates of which it is not a participant. The leaf server must be a terminal point in a replication hierarchy.

Related information

Hierarchical Routing Topology Terminology on page 52

The syscdrsend_buf Table

The **syscdrsend_buf** table contains buffers that give information about the send queue.

When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsend_buf** table, refer to Columns of the Buffer Tables on page 596.

The syscdrsend_txn Table

The **syscdrsend_txn** table contains information about the send queue.

When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsync_txn** table, refer to Columns of the Transaction Tables on page 595.

The syscdrserver Table

The **syscdrserver** table contains information about database servers declared to Enterprise Replication.

Column	Type	Description
servid	integer	Replication server ID
servername	char(128)	Database server group name
connstate	char(1)	Status of connection to this database server: • `C` = Connected • `D` = Connection disconnected (will be retried) • `T` = Idle time-out caused connection to terminate • `X` = Connection closed by user command and unavailable until reset by user
connstatechange	integer	Time that connection state was last changed
servstate	char(50)	Status of this database server:

Column	Type	Description
		- A = Active - D = Disabled - S = Suspended - Q = Quiescent (initial sync state only)
ishub	char(1)	- Y = Server is a hub - N = Server is not a hub
isleaf	char(1)	- Y = Server is a leaf server - N = Server connection is not a leaf server
rootserverid	integer	The identifier of the root server
forwardnodeid	integer	The identifier of the parent server
idletimeout	integer	Idle time-out
atsdir	lvarchar	ATS directory spooling name
risdir	lvarchar	RIS directory spooling name

The syscdrsync_buf Table

The **syscdrsync_buf** table contains buffers that give information about the synchronization queue. Enterprise Replication uses this queue only when defining a replication server and synchronizing its global catalog with another replication server.

For information on the columns of the **syscdrsync_buf** table, refer to Columns of the Buffer Tables on page 596.

The syscdrsync_txn Table

The **syscdrsync_txn** table contains information about the synchronization queue. This queue is currently used only when defining a replication server and synchronizing its global catalog with another replication server. The synchronization queue is an in-memory-only queue.

For information on the columns of the **syscdrsync_txn** table, refer to Columns of the Transaction Tables on page 595.

The syscdrtsapply table

The **syscdrtsapply** table lists statistics about the time series elements that are applied on target servers.

Table 42. The syscdrtsapply table

Column	Type	Description
source	integer	CDRID of source server.
replid	integer	Replicate ID.
txncnt	integer	The number of transactions from this source server and replicate.
tsinserts	integer	The number of time series elements that were inserted.
tsdeletes	integer	The number of time series elements that were deleted.
tscmd	integer	The number of TimeSeries routines that inserted or deleted elements that are replicated.
last_tgt_apply	integer	The time when the most recent transaction was applied on a target server.
last_src_apply	integer	The time when the most recent transaction was applied on the source server.

The syscdrtx Table

The **syscdrtx** table contains information about Enterprise Replication transactions.

Column	Type	Description
srvid	integer	Server ID
srvname	char(128)	Name of database server from which data is received
txprocssd	integer	Transaction processed from database server *srvname*
txcmmtd	integer	Transaction committed from database server *srvname*
txabrtd	integer	Transaction aborted from database server *srvname*
rowscmmtd	integer	Rows committed from database server *srvname*
rowsabrtd	integer	Rows aborted from database server *srvname*
txbadcnt	integer	Number of transactions with source commit time (on database server *srvname*) greater than target commit time

Enterprise Replication Queues

One group of **sysmaster** tables shows information about Enterprise Replication queues. The **sysmaster** database reports the status of these queues in the tables that have the suffixes **_buf** and **_txn**.

The name of each table that describes an Enterprise Replication queue is composed of the following three pieces:

- **syscdr**, which indicates that the table describes Enterprise Replication
- An abbreviation that indicates which queue the table describes
- A suffix, either **_buf** or **_txn**, which specifies whether the table includes buffers or transactions

Selecting from these tables provides information about the contents of each queue. For example, the following SELECT statement returns a list of all transactions queued on the send queue:

```
SELECT * FROM syscdrsend_txn
```

The following example returns a list of all transactions queued on the in-memory send queue and returns the number of buffers and the size of each buffer for each transaction on the send queue:

```
SELECT cbkeyserverid,cbkeyid,cbkeypos,count(*),sum(cbsize)
   from syscdrsend_buf
   group by cbkeyserverid, cbkeyid, cbkeypos
   order by cbkeyserverid, cbkeyid, cbkeypos
```

All queues are present on all the replication servers, regardless of whether the replication server is a source or a target for a particular transaction. Some of the queues are always empty. For instance, the send queue on a target-only server is always empty.

Each queue is two-dimensional. Every queue has a list of transaction headers. Each transaction header in turn has a list of buffers that belong to that transaction.

Columns of the Transaction Tables

The transaction tables contain information about transactions that are in memory. They do not contain information about transactions that are spooled to disk.

The names of transaction tables end with **_txn**. All transaction tables have the same columns and the same column definitions.

The **ctstamp1** and **ctstamp2** columns combine to form the primary key for these tables.

Column	Type	Description
ctkeyserverid	integer	Server ID of the database server where this data originated. This server ID is the group ID from the `sqlhosts` file or the SQLHOSTS registry key.This server ID is the group ID from the `sqlhosts` file.
ctkeyid	integer	Logical log ID.
ctkeypos	integer	Position in the logical log on the source server for the transaction that is represented by the buffer.
ctkeysequence	integer	Sequence number for the buffer within the transaction.

Column	Type	Description
ctstamp1	integer	Together with **ctstamp2**, forms an insertion stamp that specifies the order of the transaction in the queue.
ctstamp2	integer	Together with **ctstamp1**, forms an insertion stamp that specifies the order of the transaction in the queue.
ctcommittime	integer	Time when the transaction represented by this buffer was committed.
ctuserid	integer	Login ID of the user who committed this transaction.
ctfromid	integer	Server ID of the server that sent this transaction. Used only in hierarchical replication.

Columns of the Buffer Tables

The buffer tables contain information about the buffers that form the transactions that are listed in the transaction tables.

The names of buffer tables end with **_buf**. All buffer tables contain the same columns and the same column definitions.

Column	Type	Description
cbflags	integer	Internal flags for this buffer.
cbsize	integer	Size of this buffer in bytes.
cbkeyserverid	integer	Server ID of the database server where this data originated. This server ID is the group ID from the `sqlhosts` file or the SQLHOSTS registry key.This server ID is the group ID from the `sqlhosts` file.
cbkeyid	integer	Login ID of the user who originated the transaction that is represented by this buffer.
cbkeypos	integer	Log position on the source server for the transaction that is represented by this buffer.
cbkeysequence	integer	Sequence number for this buffer within the transaction.
cbreplid	integer	Replicate identifier for the data in this buffer.
cbcommittime	integer	Time when the transaction represented by this buffer was committed.

The following columns combine to form the primary key for this table: **cbkeyserverid**, **cbkeyid**, **cbkeypos**, **cbkeysequence**.

The columns **cbkeyserverid**, **cbkeyid**, and **cbkeypos** form a foreign key that points to the transaction in the **_txn** table that the buffer represents.

Replication Examples

This appendix contains simple examples of replication using the command-line utility (CLU).

The cdr utility on page 253 documents the CLU.

Related information

Preparing the Network Environment on page 55

Replication Example Environment

To run the replication examples in this publication, you must set up HCL® Informix® database servers. Each database server must be in a database server group.

The replication environment for the examples consists of:

- Three computers (**s1**, **s2**, and **s3**) that host the database servers **usa**, **italy**, and **japan**. Each computer has active network connections to the other two computers.
- The database servers **usa**, **italy**, and **japan** are members of the database server groups **g_usa**, **g_italy**, and **g_japan**.

Windows: For information about how to prepare the SQLHOSTS connectivity information on Windows™ systems, see SQLHOSTS Registry Key (Windows) on page 605. The following example uses the `sqlhosts` file on UNIX™ operating systems.

The `sqlhosts` file for each database server must contain the following connectivity information.

```
g_usa    group     -      -          i=1
usa      ontlitcp  s1     techpubs1  g=g_usa
g_italy  group     -      -          i=8
italy    ontlitcp  s2     techpubs2  g=g_ital
g_japan  group     -      -          i=6
japan    ontlitcp  s3     techpubs6  g=g_japan
```

You must create an sbspace for the row data and set the CDR_QDATA_SBSPACE parameter to the location of that sbspace. For more information, see Setting Up Send and Receive Queue Spool Areas on page 66 and CDR_QDATA_SBSPACE Configuration Parameter on page 512.

All commands in this example, except for creation of the sample databases on **italy** and **japan**, are issued from the computer **s1**.

The databases for the examples are identical to **stores_demo** databases with logging, as follows:

- Create a database named **stores** on the **usa** database server:

  ```
  s1> dbaccessdemo -log stores
  ```

- Create a database named **stores** on the **italy** database server:

  ```
  s1> rlogin s2
  s2> dbaccessdemo -log stores
  ```

- Create a database named **stores** on the **japan** database server:

```
s1> rlogin s3
s3> dbaccessdemo -log stores
```

For information about preparing data for replication, see Data Preparation Example on page 85.

Primary-Target Example

This is a simple example of *primary-target* replication.

In primary-target replication, only changes to the primary table are replicated to the other tables in the replicate. Changes to the secondary tables are not replicated.

In this example, define the **g_usa** and **g_italy** database server groups as Enterprise Replication servers and create a replicate, **repl1**.

To define the database server groups and create the replicate

1. Create and populate the database that defines the **usa** database server as a replication server:

   ```
   cdr define server --init g_usa
   ```

   Before replicating data, you must define the database servers as *replication servers*. A replication server is a database server that has an extra database that holds replication information.

   The **--init** option specifies that this server is a new replication server. When you define a replication server, you *must* use the name of the database server group (**g_usa**) rather than the database server name.

2. Display the replication server that you defined to verify that the definition succeeded:

   ```
   cdr list server
   ```

   The command returns the following information:

   ```
   SERVER     ID STATE     STATUS     QUEUE   CONNECTION CHANGED
   ----------------------------------------------------------
   g_usa       1 Active    Local       0
   ```

3. Define the second database server, **italy**, as a replication server:

   ```
   cdr define server --connect=italy --init \
   --sync=g_usa g_italy
   ```

   The **--connect** option allows you to define **italy** (on the **s2** computer) while working at the **s1** (**usa**) computer. The **--sync** option instructs the command to use the already-defined replication server (**g_usa**) as a pattern for the new definition. The **--sync** option also links the two replication servers into a replication environment.

   Tip: In all options except the **--connect** option, Enterprise Replication uses the name of the database server group to which a database server belongs, instead of the name of the database server itself.

4. Verify that the second definition succeeded:

   ```
   cdr list server
   ```

The command returns the following information:

```
SERVER    ID STATE    STATUS    QUEUE   CONNECTION CHANGED
------------------------------------------------------------
g_italy    8 Active   Connected 0       JUN 14 14:38:44 2000
g_usa      1 Active   Local     0
```

5. Define the replicate **repl1**:

   ```
   cdr define replicate --conflict=ignore repl1 \
   "P stores@g_usa:informix.manufact" \
   "select * from manufact" \
   "R stores@g_italy:informix.manufact" \
   "select * from manufact"
   ```

   These lines are all one command. The backslashes (\) at the end of the lines indicate that the command continues on the next line.

   This step specifies that conflicts should be ignored and describes two participants, **usa** and **italy**, in the replicate:

   - The `P` indicates that in this replicate **usa** is a primary server. That is, if any data in the selected columns changes, that changed data should be sent to the secondary servers.
   - The `R` indicates that in this replicate **italy** is a secondary server (receive-only). The specified table and columns receive information that is sent from the primary server. Changes to those columns on **italy** are *not* replicated.
6. Display the replicate that you defined, so that you can verify that the definition succeeded:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ------------------------------------------------
   REPLICATE:     repl1
   STATE:         Inactive
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.manufact
                  g_italy:informix.manufact
   ```

   Step 5 *defines* a replicate but does not make the replicate active. The output of step 6 shows that the STATE of the replicate is INACTIVE.
7. Make the replicate active:

   ```
    cdr start repl1
   ```
8. Display the replicate so that you can verify that the STATE has changed to ACTIVE:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ------------------------------------------------
   REPLICATE:     repl1
   STATE:         Active
   ```

```
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.manufact
               g_italy:informix.manufact
```

If any changes are made to the **manufact** table, the changes will be replicated to the other participants in the replicate.

Now you can modify the **manufact** table on the **usa** database server and see the change reflected in the **manufact** table on the **italy** database server.

To cause a replication

1. Use DB-Access to insert a value into the **manufact** table on **usa**:

   ```
   INSERT INTO stores@usa:manufact \
   VALUES ('AWN','Allwyn','8');
   ```

2. Observe the changes on **usa** and on **italy**:

   ```
   SELECT * from stores@usa:manufact
   SELECT * from stores@italy:manufact
   ```

In **repl1**, **usa** is the primary database server and **italy** is the target. Changes made to the **manufact** table on **italy** do not replicate to **usa**.

To not cause a replication

1. Use DB-Access to insert a value into the **manufact** table on **italy**:

   ```
   INSERT INTO stores@italy:manufact \
   VALUES ('ZZZ','Zip','9');
   ```

2. Verify that the change occurred on **italy** but did not replicate to the **manufact** table on **usa**:

   ```
   SELECT * from stores@usa:manufact
   SELECT * from stores@italy:manufact
   ```

Update-Anywhere Example

This example builds on the primary-target example and creates a simple *update-anywhere* replication.

In update-anywhere replication, changes to *any* table in the replicate are replicated to *all* other tables in the replicate. In this example, any change to the **stock** table of the **stores** database on any database server in the replicate will be replicated to the **stock** table on the other database servers.

In this example, define the **repl2** replicate.

To prepare for update-anywhere replication

1. Define the replicate, **repl2**:

   ```
   cdr define replicate --conflict=ignore repl2 \
   "stores@g_usa:informix.stock" "select * from stock" \
   "stores@g_italy:informix.stock" "select * from stock"
   ```

   These lines are all one command. The backslashes (\) at the end of the lines indicate that the command continues on the next line.

   This step specifies that conflicts should be ignored and describes two participants, **usa** and **italy** (including the table and the columns to replicate) in the replicate.

   Because neither `P` (primary) nor `R` (receive-only) is specified, the replicate is defined as update-anywhere. If any data in the selected columns changes, on either participant, that changed data should be sent to the other participants in the replicate.

2. Display all the replicates so that you can verify that your definition of **repl2** succeeded:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ---------------------------------------------------------
   REPLICATE:     repl1
   STATE:         Active
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.manufact
                  g_italy:informix.manufact

   REPLICATE:     repl2
   STATE:         Inactive
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.stock
                  g_italy:informix.manufact
   ```

   Although this output shows that **repl2** exists, it does not show the *participant modifiers* (the SELECT statements) for **repl2**.

3. Display the participant modifiers for **repl2**:

   ```
   cdr list replicate repl2
   ```

   This command returns the following information:

   ```
   REPLICATE  TABLE                         SELECT
   ------------------------------------------------------------
   repl2      stores@g_usa:informix.stock   select * from stock
   repl2      stores@g_italy:informix.stock select * from stock
   ```

4. Add the **japan** database server to the replication system already defined in the previous example:

```
cdr define server --connect=japan --init \
--sync=g_usa g_japan
```

You can use either **g_usa** or **g_italy** in the **--sync** option.

Enterprise Replication maintains identical information on all servers that participate in the replication system. Therefore, when you add the **japan** database server, information about that server is propagated to all previously-defined replication servers (**usa** and **italy**).

5. Display the replication servers so that you can verify that the definition succeeded:

   ```
   cdr list server
   ```

   The command returns the following information:

   ```
   SERVER    ID STATE    STATUS    QUEUE   CONNECTION CHANGED
   ----------------------------------------------------------
   g_italy    8 Active   Connected 0       JUN 14 14:38:44 2000
   g_japan    6 Active   Connected 0       JUN 14 14:38:44 2000
   g_usa      1 Active   Local     0
   ```

6. Add the participant and participant modifier to **repl2**:

   ```
   cdr change replicate --add repl2 \
   "stores@g_japan:informix.stock" "select * from stock"
   ```

7. Display detailed information about **repl2** after adding the participant in step 6:

   ```
   cdr list replicate repl2
   ```

   The command returns the following information:

   ```
   REPLICATE  TABLE                         SELECT
   ------------------------------------------------------------
   repl2      stores@g_usa:informix.stock   select * from stock
   repl2      stores@g_italy:informix.stock select * from stock
   repl2      stores@g_japan:informix.stock select * from stock
   ```

8. Make the replicate active:

   ```
    cdr start repl2
   ```

9. Display a list of replicates so that you can verify that the STATE of **repl2** has changed to ACTIVE:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ------------------------------------------------------
   REPLICATE:     repl1
   STATE:         Active
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.manufact
                  g_italy:informix.manufact

   REPLICATE:     repl2
   STATE:         Active
   ```

```
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.stock
               g_italy:informix.manufact
               g_japan:informix.manufact
```

Now you can modify the **stock** table on one database server and see the change reflected on the other database servers.

To cause a replication

1. Use DB-Access to insert a line into the **stock** table on **usa**:

```
INSERT INTO stores@usa:stock VALUES (401, "
PRC"
, "
ski boots"
, 200.00,
                "
pair"
, "
pair"
);
```

2. Observe the change on the **italy** and **japan** database servers:

```
SELECT * from stores@italy:stock;
SELECT * from stores@japan:stock;
```

To update the stock table on the japan database server

1. Use DB-Access to change a value in the **stock** table on **japan**:

```
UPDATE stores@japan:stock SET unit_price = 190.00
WHERE stock_num = 401;
```

2. Verify that the change has replicated to the **stock** table on **usa** and on **italy**:

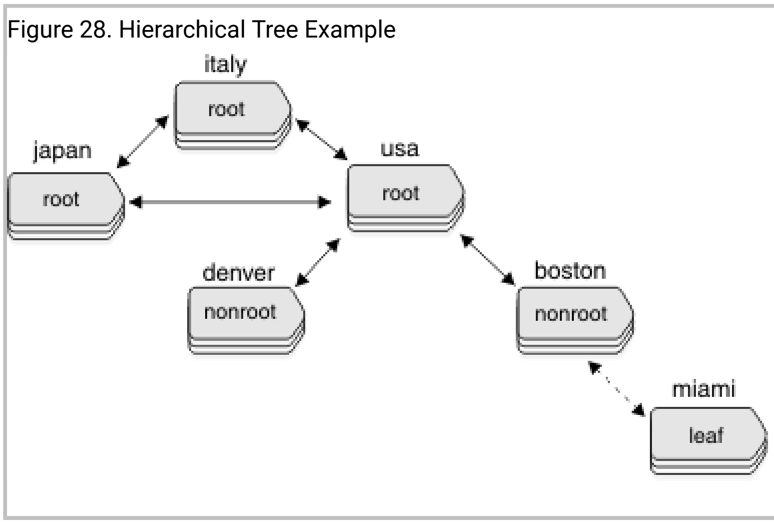
```
SELECT * from stores@usa:stock WHERE stock_num = 401;
SELECT * from stores@italy:stock WHERE stock_num = 401;
```

Hierarchy Example

This example adds a replication tree to the fully-connected environment of the **usa**, **italy**, and **japan** replication servers.

The nonroot servers **boston** and **denver** are children of **usa**. (The leaf server **miami** is a child of **boston**.) Figure 28: Hierarchical Tree Example on page 604 shows the replication hierarchy.

Figure 28. Hierarchical Tree Example



To try this example, you need to prepare three additional database servers: **boston**, **denver**, and **miami**. To prepare the database servers, use the techniques described in Replication Example Environment on page 597.

The following example defines a replication hierarchy that includes **denver**, **boston**, and **miami** and, whose root is **usa**.

To define a hierarchy

1. Add **boston** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server --connect=boston --nonroot --init \
   --sync g_usa g_boston
   ```

   The backslash (\) indicates that the command continues on the next line.

2. Add **denver** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server -c denver -I -N --ats=/ix/myats \
   -S g_usa g_denver
   ```

   This command uses short forms for the **connect**, **init**, and **sync** options. (For information about the short forms, refer to Option Abbreviations on page 254.) The command also specifies a directory for collecting information about failed replication transactions, **/ix/myats**.

3. List the replication servers as seen by the **usa** replication server:

   ```
   cdr list server
   ```

   The root server **usa** is fully connected to all the other root servers. Therefore **usa** knows the connection status of all other root servers and of its two child servers, **denver** and **boston**. The command returns the following information:

   ```
   SERVER    ID   STATE   STATUS    QUEUE   CONNECTION CHANGED
   -----------------------------------------------------------
   g_boston   3   Active  Connected    0    Aug 19 14:20:03 2000
   g_denver  27   Active  Connected    0    Aug 19 14:20:03 2000
   g_italy    8   Active  Connected    0    Aug 19 14:20:03 2000
   g_japan    6   Active  Connected    0    Aug 19 14:20:03 2000
   g_usa      1   Active  Local        0
   ```

4. List the replication servers as seen by the **denver** replication server:

   ```
   cdr list server --connect=denver
   ```

   The nonroot server **denver** has a complete global catalog of replication information, so it knows all the other servers in its replication system. However, **denver** knows the connection status only of itself and its parent, **usa**.

   The command returns the following information:

   ```
   SERVER    ID   STATE   STATUS   QUEUE   CONNECTION CHANGED
   -----------------------------------------------------------
   g_boston   3   Active              0
   g_denver  27   Active  Local       0
   g_italy    8   Active              0
   g_japan    6   Active              0
   g_usa      1   Active  Connected    0   Aug 19 14:20:03 2000
   ```

5. Define **miami** as a leaf server whose parent is **boston**:

   ```
   cdr define server -c miami -I --leaf -S g_boston g_miami
   ```

6. List the replication servers as seen by **miami**:

   ```
   cdr list server -c miami
   ```

   As a leaf replication server, **miami** has a limited catalog of replication information. It knows only about itself and its parent.

   The command returns the following information:

   ```
   SERVER    ID   STATE   STATUS   QUEUE   CONNECTION CHANGED
   -----------------------------------------------------------
   g_boston   3   Active  Connected    0   Aug19 14:35:17 2000
   g_miami    4   Active  Local       0
   ```

7. List details about the **usa** replication server:

   ```
   cdr list server g_usa
   ```

   The server is a *hub*; that is, it forwards replication information to and from other servers. It uses the default values for idle timeout, send queue, receive queue, and ATS directory.

   The command returns the following information:

   ```
   NAME       ID   ATTRIBUTES
   --------------------------------------------------------------
   g_usa      1    timeout=15 hub sendq=rootdbs recvq=rootdbs atsdir=/tmp
   ```

SQLHOSTS Registry Key (Windows™)

In an Enterprise Replication environment, you manage connectivity information about Windows™ operating systems with the SQLHOST registry key.

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

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

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
  ```
- For an Informix® 32-bit product installed on a Windows™ 64-bit system, the **sqlhosts** information is in this key:

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

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

A 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.

The Location of the SQLHOSTS Registry Key

When you install the database server on Windows™, 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

Local SQLHOSTS Registry Key

If you use the SQLHOSTS registry key on the local computer, for all database servers, the correct SQLHOSTS registry key must exist on *all* computers involved in replication. In addition, the **hosts** and **services** files on each computer must include information about all database servers.

For example, to set up replication between the server instance **srv1** on the computer **host1** and the server instance **srv2** on **host2**, you must ensure the following:

- Both **host1** and **host2** include SQLHOSTS registry key entries for **srv1** and **srv2**.
- The **services** file on both computers includes the details of the services used by both database server instances.

Shared SQLHOSTS Registry Key

If you use a shared SQLHOSTS registry key, you do not need to maintain the same **sqlhosts** information on multiple computers. However, the **hosts** and **services** files on *each* computer must contain information about all computers that have database servers.

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

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

Preparing the SQLHOSTS Connectivity Information

Preparing the SQLHOSTS connectivity information consists of setting up registry keys on each computer that hosts a database server that participates in a replicate.

To prepare the SQLHOSTS connectivity information

1. Set up the SQLHOSTS registry key for each database server on the local computer.
2. Set up the database server group registry key on the local computer.
   See Setting Up the Database Server Group Registry Key on page 608.
3. Set up the SQLHOSTS and group registry keys on all computers that are participants in the replicate.
   See Setting up the Registry Keys on All Computers on page 609.
4. Ensure that the services files on each computer include entries for all database servers that are participants in the replicate.
   See Verifying the services Files on All Computers on page 610.

Setting up the SQLHOSTS Registry with ISA

It is strongly recommended that you use IBM® Informix® Server Administrator (ISA), rather than **regedit**, to set up the SQLHOSTS registry key and database server group registry key on your Windows™ system. In addition, ISA allows you to administer your replication system from a web browser.

See the online help for ISA for details on setting up the SQLHOSTS registry key and database server group registry key.

Setting up the SQLHOSTS Registry Key for Database Server with regedit

You can set the SQLHOSTS registry key with the regedit utility.

About this task

It is recommended that you use ISA to set up the SQLHOSTS registry key.

! **Important:** Use extreme caution with regedit. If you make mistakes when editing the registry, you can damage the configurations, not only of your HCL® Informix® products, but of your other applications.

To set up SQLHOSTS with regedit:

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.
6. Select the new key that you just made (the key with the database server name) 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, or SERVICE).
   Give the OPTIONS field the value of the database server group to which this database server belongs.
8. Modify the value to add value data.
9. Repeat steps 6 through 8 for each field of the **sqlhosts** information.

Results

For example, a database server named **iris_112** might have a key under the SQLHOSTS key with the following values:

- HOST: iris
- OPTIONS: g=g_iris
- PROTOCOL: onsoctcp
- SERVICE: techpubs27

Setting Up the Database Server Group Registry Key

After you create the registry key for the database server, you must make a registry key for the database server group that includes the database server.

About this task

In this manual, each of the names of the database server groups are the database server names prefixed by **g_**. The **g_** prefix is not a requirement; it is just the convention that this manual uses.

To set up the database server group registry key

1. With the SQLHOSTS key selected, choose to add a new key.
2. 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.
3. Select the key with the database server group name that you just created and add a new string value for it.
4. Give the value the name of one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, SERVICE).
5. Add a value for the field.
   For a database server group, the **sqlhosts** information fields should have the following values:

```
HOST       -
OPTIONS    i=unique-integer-value
PROTOCOL   group
SERVICE    -
```

Each database server group must have an associated identifier value (**i=**) that is unique among all database servers in your environment. Enter a minus (-) for HOST and SERVICE to indicate that you are not assigning specific values to those fields.

6. Repeat steps 4 and 5 for the remaining fields of the **sqlhosts**
7. Select the database server group key and choose to add a key.
8. 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.
9. If you are combining Enterprise Replication with HDR, create keys for primary and secondary HDR servers under the same database server group.
10. Exit from the Registry Editor.

Results

For example, a database server group named **g_iris** could have a key under the SQLHOSTS key with the following values:

- HOST: -
- OPTIONS: i=5327
- PROTOCOL: group
- SERVICE: -

A key for the database server **iris_112** would appear both directly under SQLHOSTS and under **g_iris**:

```
SQLHOSTS
 iris_112
 g_iris
  iris_112
```

Related information

Creating sqlhost group entries for replication servers on page 58

Setting up the Registry Keys on All Computers

About this task

Now, update the registry keys on all computers that participate in replication.

To update the registry keys on all computers:

1. Set up the SQLHOSTS registry key on all computers that participate in replication.
   See Setting up the SQLHOSTS Registry Key for Database Server with regedit on page 607.
2. Set up the database server group registry key on all computers that participate in replication.
   See Setting Up the Database Server Group Registry Key on page 608.

Verifying the services Files on All Computers

About this task

On each computer that participates in replication, make sure that the **services** file contains entries for all the database servers.

To verify the services files on all computers:

1. Check the **services** file on the first host (for example, **host1**).

   The file might look like this:

   ```
   techpubs27     4599/tcp     # service for online instance denver
   techpubs28     4600/tcp     # service for online instance boston
   ```

2. Check the **services** file on the second host (for example, **host2**).

   The file should look the same as the file on **host1**:

   ```
   techpubs27     4599/tcp     # service for online instance denver
   techpubs28     4600/tcp     # service for online instance boston
   ```

Related information

Configuring ports and service names for replication servers on page 57

Data sync warning and error messages

Data sync warning and error messages describe problems with replicated transactions.

You can suppress these messages from being written to the ATS and RIS files. You cannot suppress code 0, DSROWCOMMITTED, which indicates that the row was committed, or code 1, DSEROW, which indicates that an error occurred.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter on page 515.

Table 43. Data sync warning and error messages

Warning or Error Code	Number	Description
DSEReplInsertOrder	2	Warning: Insert exception, row already exists in target table, converted to update
DSEReplUpdateOrder	3	Warning: Update exception, row does not exist in target table, converted to insert
DSEReplDeleteOrder	4	Warning: Delete exception, row does not exist in target table, saved in delete table
DSEReplInsert	5	Error: Insert aborted, row already exists in target table

Table 43. Data sync warning and error messages (continued)

Warning or Error Code	Number	Description
DSEReplUpdate	6	Error: Update aborted, row does not exist in target table
DSEReplDelete	7	Error: Delete aborted, row does not exist in target table
DSERowLength	8	Error: Length of replicated row is greater than row size in target table
DSEDbopType	9	Error: Unknown db operation
DSENoServerTimeCol	10	Error: Missing cdrserver and/or cdrtime columns in target table
DSEConflictRule	13	Error: Unknown conflict resolution rule defined
DSELostConflictRes	14	Error: Failed conflict resolution rule
DSENoServerName	15	Error: Global catalog cannot translate replicate server id to name
DSEColMap	16	Error: Unable to remap columns selected for replication
DSEColUncomp	17	Error: Invalid char/length in VARCHAR column
DSESPRetTypeOp	18	Error: Invalid data type or unknown operation returned by stored procedure
DSESPAbortRow	19	Error: Row aborted by stored procedure
DSESPSelCols	20	Error: Number of columns returned by stored procedure not equal to the number of columns in select statement
DSESPColTypeLen	21	Error: Invalid data type or length for selected columns returned by stored procedure
DSESPError	22	Error: Error returned by user's stored procedure
DSESPInternal	23	Error: Internal error (buffer too small for stored procedure arguments
DSENoMatchKeyInsert	24	Error: No matching key delete row for key insert
DSESql	25	Error: SQL error encountered
DSEIsam	26	Error: ISAM error encountered
DSELocalDReExist	27	Warning: Local delete row has been reinserted on local server
DSELocalDOddState	28	Warning: Unable to determine if the local delete row should be updated to the delete table
DSELocalDInDelTab	29	Warning: Row already exists in delete table for the given local delete row
DSEBlobOrder	30	Warning: Row failed conflict resolution rule but one or more blob columns were accepted

Table 43. Data sync warning and error messages (continued)

Warning or Error Code	Number	Description
DSEBlobSetToNull	31	Warning: One or more blob columns were set to NULL because data could not be sent
DSEBlobKeepLocal	32	Warning: One or more blob columns were not changed because data could not be sent
DSEBlobInvalidFlag	33	Error: Invalid user action defined for blob columns
DSEBlobAbortRow	34	Error: Row aborted by user's stored procedure due to unsent blobs
DSESPBlobRetOp	35	Error: Invalid action returned by user's stored procedure on blob columns
DSEReplDel	36	
DSENoUDTHeader	37	
DSENoUDTTrailer	38	
DSEStreamHandle	39	
DSEAttachUDREnv	40	
DSECdrreceiveSetup	41	
DSECdrreceiveCall	42	
DSECdrreceiveRetCode	43	cdrreceive returned error
DSECdrreceiveRetGarbage	44	cdrreceive returned garbage
DSEStream	45	Error reading from stream
DSEStreamAborted	46	Stream aborted by sender
DSEValStore	47	
DSECdrreceiveRetType	48	cdrreceive returned wrong type
DSEStreamOptType	49	Unrecognized stream option
DSEStreamOptLen	50	Stream option has bad length
DSEStreamOptBitmap	51	Error in changed col bitmap
DSEUnStreamColl	52	Error while unstreaming collection
DSEUnStreamRowType	53	Error while unstreaming rowtype
DSEStreamFormat	54	Unexpected or invalid data in stream
DSEStack	55	Out of stack space

Table 43. Data sync warning and error messages (continued)

Warning or Error Code	Number	Description
DSEInternal	56	Generic internal problem
DSESmartBlobCreate	57	Error creating sblob
DSESmartBlobWrite	58	Error writing sblob
DSEStreamColConv	59	Error converting column data from the master dictionary formats to the local dictionary format
DSE2UTF8CodeSetConvErr	63	Error converting data from local database code set to UTF-8
DSEFromUTF8CodeSetConvErr	64	Error converting data from UTF-8 to local database code set.
DSE2UTF8CodeSetConvWarn	65	One or more characters were substituted during conversion from the local database code set to UTF-8.
DSEFromUTF8CodeSetConvWarn	66	One or more characters were substituted during conversion from UTF-8 to the local database code set.
DSETSSetup	67	Failed to setup environment for processing time series elements.
DSETSDelOp	68	Failed to apply time series delete statement. Statements include DelClip(), DelRange(), and DelTrim() operations.
DSETSElem	69	Failed to apply time series element.
DSEOpenTSCon	133	Failed to open time series container.

Index