IBM® Information Management Software

IBM Informix Flexible Grid
Extending Data Availability

Dynamic configuration and faster performance with Flexible Grid

Less and easier administration and higher availability

Load balancing and failover strategies

Whei-Jen Chen
Venkatesh Gopal
Holger Kirstein
Madison Pruet
Nilesh Ozarkar
Ammu Soni
Amit Vashishtha

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Informix Flexible Grid: Extending Data Availability

June 2011

SG24-7937-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (June 2011)

This edition applies to IBM Informix Version 11.7.0.

Contents

Notices

Trademarks

Preface

IBM® Informix® is an exceptional online transaction processing (OLTP) database management system that offers outstanding performance, reliability, scalability, and manageability for enterprise and workgroup computing. As a full-featured relational database management system platform, Informix offers extensive capabilities to meet unique business requirements with the speed of native database functions.

These functions include enhanced performance, availability, and database administrator and programmer productivity. In addition, there is reduced deployment and management costs. As examples, you can customize the data server footprint with the Deployment Wizard. The SQL API and scheduler make it even easier to automate maintenance activities. And, the GUI-based OpenAdmin Tool for Informix database server provides a global view of remote servers, with flexible analysis and drill down to the query level.

In this IBM Redbooks® publication, we focus on, and provide an overview of the high availability and enterprise replication features of Informix 11.70. IBM Informix provides solutions for making data highly available through the MACH11 cluster. The components of the MACH11 cluster include High Availability Data Replication (HDR), Shared Disk Secondary (SDS), and Remote Secondary Standby (RSS) servers. Enterprise Replication (ER) provides a means of selectively replicating data between systems in near real time. The Informix Flexible Grid eliminates the administrative complexity of ER, and it provides the ability to automatically create database objects, such as tables, indexes, and stored procedures, on all nodes within the grid as a single operation. These enhanced enterprise replication features provide solutions for those customers requiring reliable and quick dissemination of data across a global organization.

There is also enhanced capability for customers requiring high-availability disaster recovery, in the form of the ability to resend primary server indexes to secondary servers without requiring a rebuild of the index on the primary server.

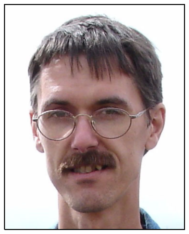
Capabilities such as these enable faster, easier, and more reliable distribution and high availability of data, resulting in improved access and use throughout the enterprise.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and IBM DB2® system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development, and an IBM Certified IT Specialist.

Venkatesh Gopal is a Senior Technical Staff Member (STSM) in the Information Management group. He has worked in Informix database development since 1996 in various areas, with a focus on client and application development technologies. He has also worked on the IBM Optim™ tools as an architect with a focus on the data life cycle tools. He currently oversees the virtualization and cloud strategy for Informix, helps with feature development in these areas, and also sees how Informix can align with the broader IBM virtualization and cloud strategy.



Holger Kirstein is a resolution team engineer with the European Informix support team. He joined the Informix support team in 1996 and has over 15 years experience in application development and support for Informix database servers and Informix clients. He holds a Masters of Applied Computer Science from Technische Universität, Dresden.

Madison Pruet is the technical lead for the IBM Informix Availability Team responsible for architectural design and is the technical team leader for development of enhancements to Backup/Restore, High Availability Data Replication, and Enterprise Replication for Informix. He joined IBM as part of the Informix acquisition and joined Informix in the early 1990s. Prior to joining Informix, Madison was DBA and Systems Manager for La Quinta Inns. Earlier positions were with Spery-Univac and Xerox.

Nilesh Ozarkar is an Advisory Software Engineer in the IBM Information Management Division, in Lenexa, Kansas. After joining IBM Informix in 2000, he worked on several Informix components, participated in beta programs, and presented at technical conferences. In recent years, his primary focus has been on Informix cluster and flexible grid replication technology. Before joining IBM, he worked in the payment processing industry, providing technical support for a card management system. Nilesh holds a Bachelor's degree in Electronics and Telecommunication from Amravati University, India.

Ammu Soni is an Advisory Software Engineer in IBM Information Management Division. Ammu joined Informix Software in 1997 and later joined IBM as part of the IBM acquisition of Informix. She works system testing for various Informix components across various platforms, with a focus on Backup and Restore and High Availability. Ammu has a Bachelor's degree in Computer Science.

Amit Vashishtha is an Advisory Software Engineer with the IBM Informix development team in San Jose, California. He has worked across various Informix server engineering areas for more than a decade. He has been a member of the Backup and Restore, High-Availability, Embeddability and Cloud Computing teams in recent years. Amit is also an IBM Certified Solution Architect for Cloud Computing Infrastructure V1 and IBM Certified Solution Advisor for Cloud Computing Architecture V1.

Acknowledgements

Thanks to the following people for their contributions to this project:

Mahesh Dashora
Bill Belisle
Dick Snoke
Donald Hutchinson
IBM Software Group, USA

Mary Comianos, Operations and Communications
Ann Lund, Residency Administration
Emma Jacobs, Graphics
International Technical Support Organization, San Jose Center

Thanks to the authors of *Informix Dynamic Server 11: Extending Availability and Replication*, SG24-7488:

Chuck Ballard
Joseph Baric
Ajay Gupta
Holger Kirstein
Rajesh Naicken
Nilesh Ozarkar

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1 Overview

Data availability is a critical requirement of our day to day lives. It is important to not only ensure that the data is available when we need it, but to also that it is available *where* we need it. If the data becomes unavailable, then the business process can come to a halt. Customers cannot pay for their purchases because the payments cannot be posted. Orders cannot be taken. Inventory cannot be processed. Medical records cannot be examined. Policemen cannot check their records to determine if a simple traffic stop involves a fugitive from justice. And the list goes on. It is critical that data be available when and where we need it.

Additionally, it is important to have enough resources to process the data when it is needed. We expect to get a return from a web browser in less that six seconds. If it takes too long to access or process our data, then we give up and go elsewhere. If we want to remain completive, then we must be able to expand the available resources to process that data based on usage demands. As an example, during the pre-Christmas season, a website for a retail store probably need to have more resources than it does during other times of the year.

IBM Informix provides solutions for making data highly available through the MACH11 cluster. The components of the MACH11 cluster include High Availability Data Replication (HDR), Shared Disk Secondary (SDS), and Remote Secondary Standby (RSS) servers. While each of these components have some specific differences, they all provide the ability to create a highly available environment. In addition, they all provide increased capacity as they all support transactional usage of the redundant nodes.

In addition to making data scalable and highly available, we also must make data available *where* we need it. As an example, consider a website for a retail company (Figure 1-1).

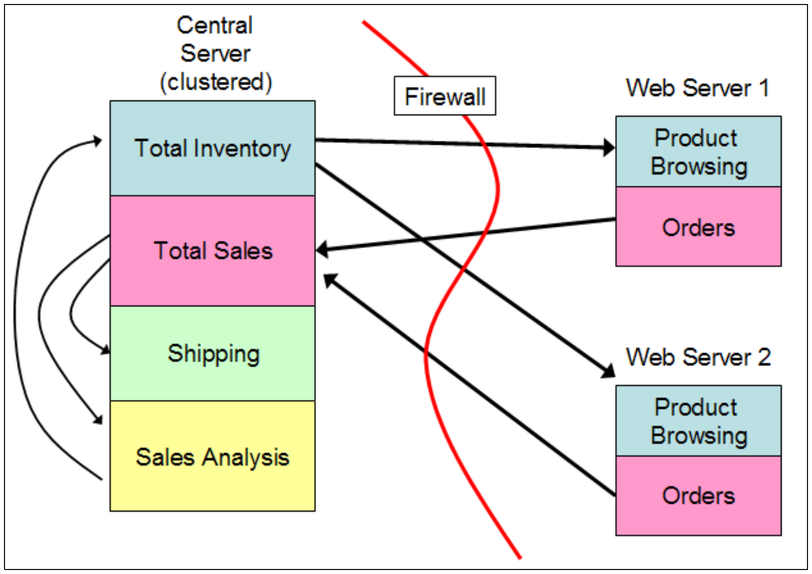


Figure 1-1 Example company website

Their central database uses a clustering solution to provide high availability. The web server uses a separate database that contains products available for sale and is also used to take orders. When a customer places an order, that information is replicated to the central database. Analysis is done on the central server to determine if purchase trends are developing and when a trend is uncovered, then the inventory on the web server is adjusted, perhaps to change the products displayed on the home page, to adjust the ordering of items as they are displayed, or possibility to change the inventory on the web server. All of these actions are done to maximize sales.

The company has two web servers that not only provides availability in the event that one of the web servers goes down or is overloaded, but also to reflect regional differences in purchase trends. Although each is a failover server for the other, web server 1 has a preference for cold weather climate items and web server 2 has a preference for warm weather climate items.

We have met the requirement of availability by using a clustering solution for the central server and have redundant web servers. We have met the requirement of having additional resources as needed because the alternate web server can be used in the event that the other is overloaded. Also, we have addressed the issue of making data available where we need it.

Enterprise Replication (ER) provides a means of selectively replicating data between systems in near real time. While ER is rich in functionality, that richness also increases the administration complexity. Informix has tried to continually reduce the complexity of ER by introducing functionality such as replication templates and replsets.

The IBM Informix Flexible Grid eliminates the administrative complexity of ER. The Flexible Grid provides the ability to automatically create database objects such as tables, indexes, and stored procedures on all nodes within the grid as a single operation. Additionally, the Flexible Grid can automatically establish near real-time synchronization of data across those nodes. If your business has to expand capacity, a new server can be added to the grid in minimal time and will be synchronized with the existing grid.

However, it does little good if our data is highly available or available in multiple locations but our application does not know how to access that data. The Connection Manager (CM) has knowledge about the MACH11 cluster and of the Flexible Grid. By using CM, the application does not have to worry about where the data is, or which server is the best location to use. The Connection Manager provides a network virtualization layer that allows the application to simply connect to the best location to access your data. The underlying physical systems might change, but the client application does not have to know about those changes.

1.1 Clusters

A *cluster* is when there are multiple systems that work as a unit. Clusters are generally used to provide high availability (HA), and to scale out the database for increased performance. If used for high availability and the desired goal is no down time, then there must be additional considerations to ensure that there is no single point of failure (SPoF). This means that there must be redundancy for anything that could result in a failure of the system. The processors must be redundant, the disk must have redundancy, the network must have redundancy, and so on.

Generally, the higher the degree of availability, the more expensive the solution. For many people, RAID 5 is considered to be enough for disk availability, but that still results in a potential failure because the disk controller can fail. Disk mirroring might provide a better degree of high availability if the disks are on different controllers, but that solution costs more. Also, there is the problem of the environment. If the disks are in the same location, then they are vulnerable to fire, flood, or other disasters. So a higher degree of availability can be provided by making the redundant disk physically removed from each other.

There is one primary and multiple secondary servers that make up the MACH11 cluster. The secondary servers can consist of a single HDR secondary server, one or more Shared Disk Secondary servers (SDS), and one or more Remote Secondary Standby (RSS). The MACH11 cluster can consist of any combination of secondary nodes. Although there may be some minor differences in each of these secondary types, they all provide multiple levels of failover as well as multiple servers for scale-out.

The MACH11 cluster consists of any combination of HDR, RSS, and SDS. For instance, we can have an HDR and an RSS server in a cluster. Furthermore, the RSS can be in a remote location, such as a bunker site, or maybe in a public cloud, so that we can have an offsite secondary while keeping the HDR secondary onsite. This way, we have an easy way to manage a quick failover solution for most situations, but also provide for recovery if a more catastrophic failure should occur. Figure 1-2 demonstrates this solution. If the primary should fail, then the HDR secondary can be promoted to primary. At that point, it automatically connects to the RSS server and continues to ship the logs to the RSS server.

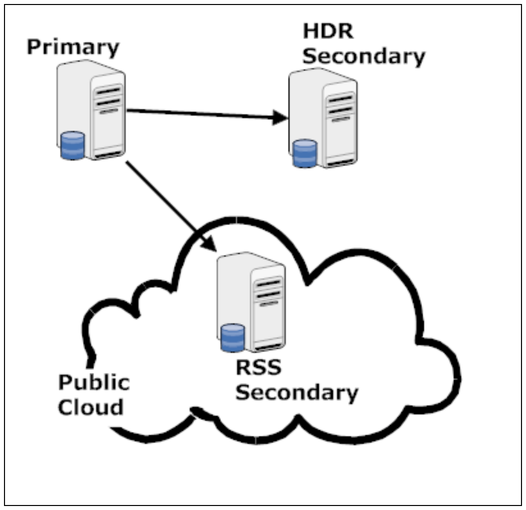


Figure 1-2 Cluster with HDR and RSS

1.2 Enterprise Replication

Enterprise Replication (ER) provides the ability to replicate data to and from multiple database systems. It has rich functionality, supporting full row replication, selective replication, update anywhere, and conflict resolution. We can replicate part of a table to multiple nodes where some rows go to some targets and other nodes can go to another target. We often refer to all of the servers acting as a source or target of ER as being a member of the ER domain.

Each server in the ER domain is an independent server, which is different than the secondary nodes of the MACH11 cluster. Each server can have data that is local to that server as well as data that is common across the ER domain.

Additionally, each node within the ER domain can be a MACH11 cluster. This means that is possible to have a cluster of clusters with near limitless scale-out.

The rich feature set of ER does come with a price. Administration can be a bit daunting. Informix has reduced the complexity over the years by introducing such functionality as templates, and has reduced the administrative requirements even further with the Flexible Grid.

1.3 The Flexible Grid

The Flexible Grid allows us to have multiple servers on which everything is replicated, not just the data changes made to tables. Inserts, updates, and deletions are replicated, but so are statements such as “create database...”, “create table...”, “create index....”, “create procedure...”, “grant...”. Because data definition language (DDL) statements are replicated as well as data manipulation language (DML), each of the nodes within the Flexible Grid can be fully used as an independent server, providing for near limitless scale-out.

The Flexible Grid may be built on ER, but only requires a small subset of the ER administration, that is, the ER commands to define how the servers are connected to each other to perform propagation. You do not have to define what is to be replicated or where it is to be replicated. You simply issue the “create table...” statement, and that table is created on all nodes within the Flexible Grid and data synchronization is automatically established.

You can dynamically add nodes to the Flexible Grid based on business requirements. Perhaps you must add a new node to provide additional resources for a new website. You can use the ifxclone utility to add the new node. After cloning, the new node is automatically included as a member of the Flexible Grid.

1.4 The Connection Manager

Although we increase the availability of data by having data located in multiple locations, we also introduce more complexity. How do we know which copy of the data that we should process? For example, we might not need to have the most current copy of the data if we are running statistical reports, In that case, we can probably use a secondary server, possibly an SDS server. If we are performing a massive update or have many active updates, however, we probably would want to perform that on the primary server.

An additional problem is that we connect typically to a server based on the server name or possibly the IP address and port number. But what do we do if the nature of the server changes? Perhaps it is important to ensure that the application always run on the primary node. But exactly where is the current primary?

Also, consider ER. We might be able to run on any of the nodes to which that data is replicated, but we want to ensure that we are not executing on a server in which the apply has a high latency or has had a significant number of apply failures. We might not consider the quality of that data to be good enough to process.

Finally, we want to try to equalize the workload. If we have the data on multiple machines and one of those machines is fairly idle, then it only makes sense to process on that machine.

The Connection Manager addresses these issues. When we use the Connection Manager, we describe a Service Level Agreement (SLA) that describes the type of server to which we want to connect. The SLA also includes other user requirements, such as data quality, workload requirements, latency, location, and a set of data that we want to process. When we connect, instead of connecting to a server, we might request a connection where the data quality is good, but the latency does not have to be that good, and we can accept a connection to a secondary server. The Connection Manager keeps track of the data quality, the latency, the workload, the server type, and other statistics. When we request a connection, then the Connection Manager redirects the connection request to the best choice based on the SLA requirements.

An example of the usage of the Connection Manager is shown in Figure 1-3.

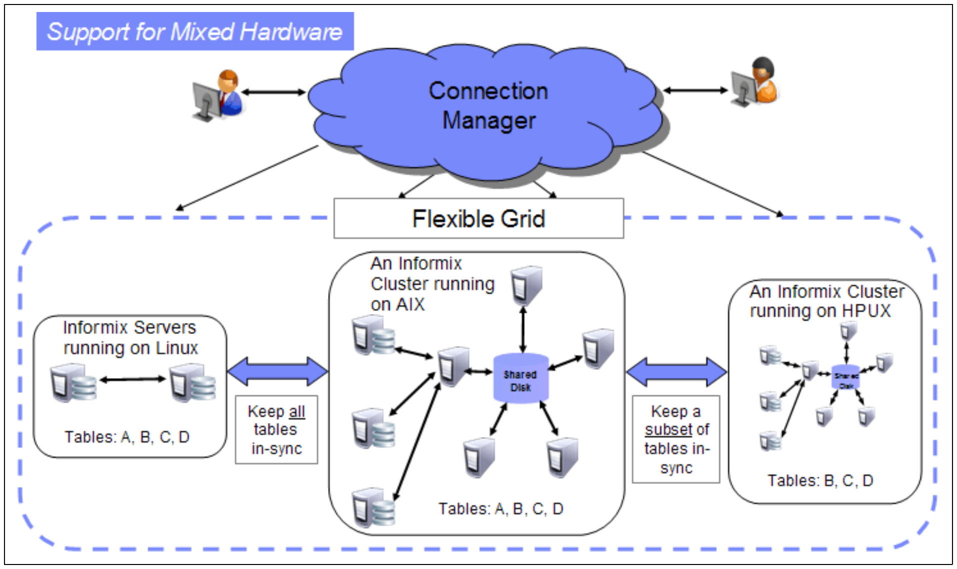

Support for Mixed Hardware
Connection Manager
Flexible Grid
Informix Servers running on Linux
Tables: A, B, C, D
Keep all tables in-sync
An Informix Cluster running on AIX
Shared Disk
Tables: A, B, C, D
Keep a subset of tables in-sync
An Informix Cluster running on HPUX
Tables: B, C, D

Figure 1-3 Connection Manager with the Flexible Grid

In this example, we have a Connection Manager that is managing the connections to servers within a Flexible Grid. It is constantly monitoring the states of the machines within that Grid and determining where it should direct a client connection request. It is worth noting that some of the servers within the Flexible Grid or single standard servers and others are actually a cluster. All of these components can be managed from a single Connection Manager, reducing the complexity of administration.

1.5 Choosing a replication strategy

The choice of a replication solution is generally driven by a business requirement to make data more available. Likewise, the choice of which replication solution is chosen is also driven by business requirements. Let us consider some basic business requirements and possible solutions:

- *Each transaction that is performed must exist on a secondary server in the event of a system failure.*

  If this is a requirement, then using HDR is probably the best solution. When used in SYNC mode and with unbuffered logging, we do not allow the commit of the transaction on the primary to complete until it has been sent to the secondary. This ensures that the committed transaction is immediately propagated to the secondary server and is available in the event of a failover.

- *I need to make my data available for massive analysis just as soon as it is loaded. I need to be able to have significant scale-out and do not want to have to pay for additional hardware.*

  In this case, SDS would be a good choice. SDS provides for additional scale-out and does not require additional disks. However, SDS in itself does not provide for failover in the event of a disk failure; the disks need to be mirrored. An alternative to disk mirroring might be using an HDR secondary in addition to SDS.

- *In addition to making my system highly available, I need to ensure that my system will survive multiple levels of failure. I need additional assurance than a single secondary server provides.*

  In this case, we might consider an HDR and RSS combination. The HDR system could provide immediate failover functionality and the RSS server provides a more offsite redundant server that is available if there is a wide-spread disaster.

- *I have a central system and need to replicate only a subset of my data to the regional offices of my company.*

  In this case, we would probably want to use Enterprise Replication. Each of the machines would function as an independent system with some data being common and some data being local to that node. If we have a further need to make the central system to be highly available, then we might consider letting it have an HDR secondary as well.

- *I need to have multiple servers for load balancing spread across a wide geographic area and also expect to have a high degree of schema changes.*

  In this case, we probably want to use the Informix Flexible Grid. Because the Flexible Grid propagates DDL changes and also automatically establishes DML replication, it would probably prove to be a good solution.

- *I can never have planned down time, even for a server upgrade.*

  Again, the best solution is probably an ER or Flexible Grid solution. ER has always supported replication across dissimilar versions of the Informix database server.

  We now have the ability to seamlessly convert an HDR or RSS server into an ER server. This function enables us to normally run as an primary + HDR secondary and then dynamically convert the secondary into an ER server. By using this setup, we can perform a rolling upgrade. After the upgrade is completed, we can re-establish the HDR (or RSS) secondary by using the ifxclone utility.

1.6 Informix and virtualization

Virtualization, as the word indicates, provides a virtual front to something actual. There are several virtualization technologies in play today, such as desktop, server, network, or storage virtualization. Server virtualization allows for reuse of an existing physical server by adding a layer on top of it, which then makes it possible to run several virtual environments. This setup allows IT organizations to effectively use their server infrastructure and allow multiple application sets with different operating system requirements to run on a single physical hardware rather than requiring separate physical servers with operating systems on each server to run the application sets. This setup leads to consolidation of resources in the IT department, less idle time on large resources, and low management and utility costs.

Server virtualization technologies can be classified broadly into two categories based on where they fit in and how they actually work:

- Type 1

  This type of virtualization is also called “bare metal” virtualization, where the virtualization layer runs directly on top of the physical hardware and divides the server machine into several virtual machines or partitions with a guest operating system running in each of the machines. This virtualization approach provides binary transparency because the virtualization environment products themselves provide transparency to the operating systems, applications, and middleware that operate above it.

Examples of this virtualization technology include IBM system logical partitions (LPARs), Linux Kernel-based Virtual Machine (KVM), Oracle VM Server for SPARC, HP Integrity Virtual Machines, Citrix XenServer, Red Hat Xen and KVM, Novell SUSE Xen, Microsoft Hyper-V, and VMware ESX Server.

The Type 1 virtualization can further be classified into three categories:

- Full virtualization
- Para virtualization
- Hardware-assisted virtualization

The difference between these three lies in how the operating system and kernel level calls made on the host are executed on the CPU.

- Type 2

  This type of virtualization is called *OS virtualization*. In this type, the virtualization environments are created with a single instance of the operating system. The virtual environments created by OS virtualization are called *containers*. Because all virtualization environments must share the resources of a single operating system while having a private virtual operating system environment, a particular implementation of the technology may alter file system orientation and often introduce access restrictions to global system configuration or settings. For example, IBM AIX® WPARs and Solaris Zones separate `/usr` and `/root` where `/usr` is shared and `/root` is private to each virtualization environment; WPARs and Zones also have restrictions so that kernel extensions, device drivers, and system clock cannot be altered within a virtualized environment. As a result, binary transparency for applications may be compromised due to the constraints.

Examples of OS-based virtualization include IBM AIX WPARs, Sun/Oracle Solaris Zones, and Parallels Virtuozzo Containers. Figure 1-4 illustrates the virtualization types.

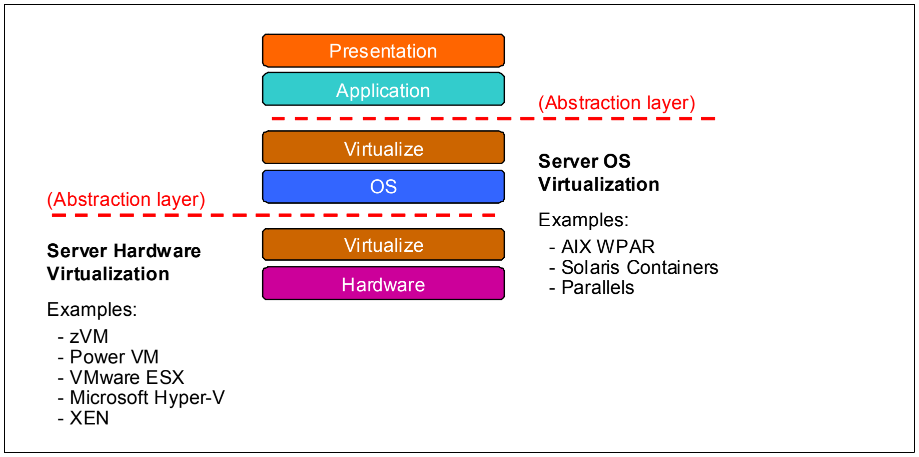


Figure 1-4 Virtualization types

Virtualization is now heavily used in data centers. Most of the existing hardware is subject to some type of virtualization to obtain maximum use out of them.

1.6.1 Virtualization platforms

There are several popular vendors that provide virtualization technologies. We discuss a few of them here:

- IBM PowerVM™

  IBM PowerVM is a virtualization platform that runs virtualized AIX, IBM i, and Linux environments on IBM POWER® processor based systems. PowerVM provides customers with the ability to create advanced logical partitions such as Micro-Partitions (that can be as small as 0.1 core per partition). Other PowerVM technologies include Virtual I/O Server (VIOS), Live Partition Mobility, and Multiple Shared-Processor Pools. For more information, go to the following address:

  http://www-03.ibm.com/systems/power/software/virtualization/

- VMware VSphere, ESX and ESXi

  VMware VSphere is a full enterprise virtualization solution that includes server, network, and storage virtualization technologies. VMware ESX and ESXi are server virtualization technologies. For more information, go to the following address:

  http://www.vmware.com

- Microsoft Hyper-V

  Microsoft Hyper-V is a virtualization platform from Microsoft that runs on Windows Server 2008. It can in turn run virtualized Linux and Windows environments. The individual virtual environments are called *partitions*. One of the partitions, called the root partition, has to be a Windows partition. For more information, go to the following address:

  http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx

- HP Integrity Virtual Machines (VM)

  HP Integrity Virtual Machines (VM) is a software virtualization technology within the HP Virtual Server Environment (HP VSE) that enables you to create multiple virtual servers or machines each with its own “guest” operating system instance, applications, and users. It supports HP 11i v2 and v3, Windows for Itanium, and Linux for Itanium guests. For more information, go to the following address:

  http://h20338.www2.hp.com/enterprise/us/en/os/hpux11i-partitioning-integrity-vm.html

- Linux KVM

  Linux Kernel-based Virtual Machine (KVM) is a virtualization technology that is built into the Linux kernel. This technology enables it to host guest operating systems as well as run native applications in that kernel. For more information, go to the following address:

  http://www.linux-kvm.org/page/Main_Page

- Solaris Zones

  Solaris Zones is an OS virtualization technology that is part of the Solaris 10 release. It has a global zone with a number of zones running within it. Each of these zones have their own node name, virtual network interface, and storage associated with it. For more details, go to the following address:

  http://www.oracle.com/technetwork/systems/containers/index-jsp-140778.html

1.6.2 Virtual appliances

A virtual appliance (VA) is a pre-tested and configured virtual disk image designed to run on a virtual machine. A virtual appliance usually includes applications and an operating system to run on a virtualization platform (for example, VMware, Microsoft Hyper-V, and LPAR). Virtual appliances are generally useful for quick development, test, and demo scenarios, as everything comes pre-packaged and pre-configured.

Informix 11.7 is distributed as a virtual appliance that is a VMware image and can be run on virtualization systems that support the image format. The virtual appliance is distributed in VMware Fusion, Workstation, and Player format, and includes the following components:

- SLES 11
- IBM Informix database software 11.70.xC1
- IBM Informix Client Software Development Kit (CSDK) 3.70.xC1
- IBM Informix JDBC Driver 3.70.xC1
- IBM Informix Spatial Datablade Module 8.21.xC4
- IBM Informix Web DataBlade® Module 4.13.xC4
- IBM Data Server Client Drivers v95fp5
- IBM OpenAdmin Tool (OAT) 2.70
- AGS Server Studio & Sentinel 7.5

You can download the Informix virtual machine image at the following address:

https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-informixfpd&S_PKG=dl

We discuss the details about how to use this image in Chapter 12, "Informix in cloud and virtualization environments" on page 461.

1.6.3 Informix support for virtualization

Database servers are essential components in an application system architecture and are normally required to run on virtual machines on the virtualization platforms. The expectation for the database systems are that the database software is certified in the guest operating systems that the virtualization platforms allow for, works in tandem with the virtualization technologies, and can benefit from certain aspects of virtualization, such as motion technologies.

Informix is certified on the guest operating systems for the different virtualization environments. Details about the supported platforms are listed in Chapter 12, “Informix in cloud and virtualization environments” on page 461. There is sub-capacity based licensing that can be applied to virtualization environments where a virtual machine with Informix can use only part of the capacity of the entire system.

In addition, Informix has features and functions that complement the virtualization capabilities and fit well into virtualization environments. We highlight a few here:

- Informix MACH11 technology

  This feature provides high availability and works on homogeneous environments. As virtualization provides homogeneity, MACH11 is well suited in these scenarios.

- Informix Flexible Grid

  This is a high availability solution for platform independent systems and can be easily set up on the disparate virtualization solutions.

- Connection Manager (CM)

  This feature provides load balancing capability and is well suited in virtualization platforms where using data center resources optimally is the key. The load on the database servers that the Connection Manager manages is distributed to match the service level agreements (SLAs) defined with the Connection Manager.

- ifxclone

  To handle additional workloads, you might need to add new nodes to your existing Informix topology. In the scale-out scenarios, having the new resource up and running with less management and administration is a basic requirement. The ifxclone utility of Informix server allows you to create new data server nodes with ease. Virtualization technologies provide easy scale-out capabilities in provisioning new guest operating systems and ifxclone complements these technologies at the data server level. You can use both of them together to set up a new cloned database environment setup even quicker.

1.7 Informix and cloud computing

Cloud computing is a computing style in which the dynamically scalable and, often, virtualized resources are provided as a service over the Internet or an intranet. A user can request a resource, either an infrastructure or software, and pay only for the resource usage. For example, if infrastructure is being requested, then you pay for the amount of computing resources that are used; if an application service, such as using database, is requested, then you pay for the database size that is being requested. The users receive all of the required capabilities, including high availability, scalability, security, and other quality of service (QoS) enhancements, but do not have to manage the infrastructure, applications, and services, or need to invest in the data center setup.

Cloud services are categorized based on the types of services provided:

- Infrastructure as a Service (IaaS): Here the cloud provides an infrastructure that the user cannot or does not want to own. Users can request computing resources, storage, and network resources. They pay for the amount of resource requested for the duration they use the resource. The users give up the resource after they are done using it. The IaaS provider manages and maintains the resources, but does not care what the resource is being used for, as long as the usage is complied with the service level agreements (SLAs). The advantage with this model is that the user can request and receive quickly the additional computing resource needed. Popular IaaS providers include IBM, Amazon, and other vendors.
- Platform as a Service (PaaS): This cloud service provides the complete environment to build and run a particular type of application, for example, a web server and database system. The user does not have to worry about provisioning, setting up the software stack, patching, and upgrades (though customizations are possible). Popular PaaS providers are SalesForce, Amazon, and Microsoft.
- Software as a Service (SaaS): Using this could service, users can work directly on the software they buy from the cloud rather than having to install and use it on their premises. Advantages of this model are that patches and upgrades are automatically handled by the service provider and the users do not have to worry about maintenance issues. Popular SaaS providers are SalesForce and Microsoft.

1.7.1 Types of cloud

The cloud environments are normally classified into three categories based on where they exist:

- On-premise: This model, also called the *private cloud*, is within the firewall managed by the IT department. Because the private clouds are within an enterprise firewall, they are more secure and are subject to corporate governance policies.
- Off-premise: This model is generally managed by an external service provider and is also called a *public cloud*.
- Mixed: The mixed model, also called a *hybrid cloud*, has resources allocated from within the private cloud and the public cloud.

The delivery model in all of these cloud types are effectively the same: Request the resource, get as much as you need, scale up and down based on SLAs provided by the user, and then give up the resource use after your work is done.

Figure 1-5 illustrates these three cloud models.

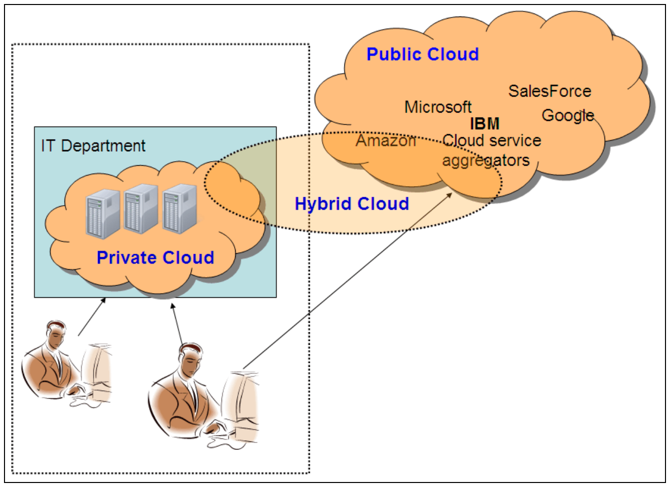


Figure 1-5 Cloud types

1.7.2 Informix in the cloud environment

Databases are used in several contexts in a cloud model, including:

- The clouds from vendors and partners offer SaaS services with a database under the covers. Several Informix partners build cloud solutions that embed Informix.

- Database as a service offerings can be made with databases where a user requests database use (specifying the size and some other parameters) and receives the connection information for the database to work with.
- Often times, users do not have the infrastructure on their premises to create quick test and development environments. Public cloud environments provide the infrastructure and the software catalog necessary to perform certain activities.
- Cloud environments can be used for additional capacity. Users can use the cloud to start additional database nodes for high availability or backup database information into the cloud (storage is allocated in the cloud rather than having to be on the user's premises).

IBM Informix offers capabilities designed for the cloud environments as well as server enhancements that can help in cloud installation:

- Informix software can be provisioned on both IBM and Amazon cloud environments. Refer to Chapter 12, "Informix in cloud and virtualization environments" on page 461 for more details.
- In Amazon S3 environments, you can back up data to the Amazon S3 storage using the **`ontape`** command with some parameters set in the `onconfig` file. Refer to Chapter 12, "Informix in cloud and virtualization environments" on page 461 for more details.
- In cloud environments, one of the requirements is less management and administration impact. Informix features such as automatic statistics management and automatic storage provisioning reduce management impact.
- In cloud environments, it may be necessary for customers to create template based configurations (preset images with customized server software, data, and configurations). The Informix deployment assistant utility helps achieve this purpose.
- In cloud environments, everything comes with a cost, including storage. Informix Data Compression can help reduce the cost of storage.
- Security is a key requirement in cloud environments, especially communication in a public cloud environment. Support for encryption and SSL in server communication helps in such scenarios.
- Multi-tenancy is a need in cloud settings where a single database table can host multiple user data. Data belonging to one user should not be seen by another. Label Based Access Control (LBAC) can help in this scenario.
- All other capabilities, including MACH11, Flexible Grid, Connection Manager, and so on, that are described in 1.6.3, "Informix support for virtualization" on page 13 apply to cloud environments as well.

In conclusion, Informix has many features that make it a good candidate for cloud environments. Also, it can be used for any kind of cloud offering, such as Software as a Service, Database as a Service, or to create customized cloud environments.

2

OpenAdmin Tool

The OpenAdmin Tool (OAT) can manage all of your production databases, including Stand-alone Servers, Enterprise Replication servers, Flexible Grid, and MACH 11 clusters. You can use OAT to perform storage management, task scheduling, utilization and performance management, cluster management, cluster and database server monitoring. OAT provides a graphical user interface that is used for monitoring. You also can use a command-line interface to use the onstat monitoring utility.

OAT is a simple to install and easy to use PHP-based application. You only need a web browser to use it, and it can be installed on one single machine, either Windows or UNIX based. From that single point, you can mange all of your servers without any additional installations on the remote systems.

In this chapter, we describe the content of the OAT package, the installation procedure, and how to set up the access to your database servers.

2.1 Basic functionality

The Informix database servers can be maintained with little effort in terms of administration resources. There are only a few required configuration and log files for each database server. After a server is running and configured according the application's needs, the configuration files require only infrequent changes.

Ongoing monitoring efforts in terms of database resources, such as memory, CPU utilization and workload, I/O throughput, and disk space management, are relatively easy and can be done with few tools. Still, they provide an all-embracing view of the database server status to the administrator.

However, database server infrastructures tend to grow, and are becoming increasingly sophisticated. There are cluster environments that have multiple servers in the same cluster with sub-clusters defined. In addition, there are database server instances distributed remotely, yet the administration department is concentrated at one location. These requirements demonstrate the need for a single point of administration for all the database servers in the production environment.

There is also a requirement for a graphics-based remote administration console. This console should provide the ability to handle the execution of all major administration tasks, such as database storage management, server restart, and the monitoring of all necessary key resources in the database server, such as log files, memory and CPU, and I/O utilization. Ideally, the server should periodically execute these monitoring tasks automatically, and when specific limits have been reached, notify the administrator by email or by raising an alarm that is visible on the console.

The OAT utility meets most of these local or remote database administration requirements. It is an open source product hosted by IBM and based on PHP scripts. It requires the existence of a web server on the machine where it is installed, which does not itself need to be the database server machine. The application runs in a standard web browser and is OS independent. Using an open source product gives you the additional capability to apply some resource monitoring windows required by your specific environment that might not currently be generally available.

In this book, we focus on the monitoring and maintenance features that OAT provides for the IBM Informix MACH11 cluster, ER, and Flexible Grid environments. We start our discussion about OAT by discussing the installation and setup of OAT.

However, OAT provides much more functionality than we can cover in this book here; for more detailed information, refer to *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-74655.

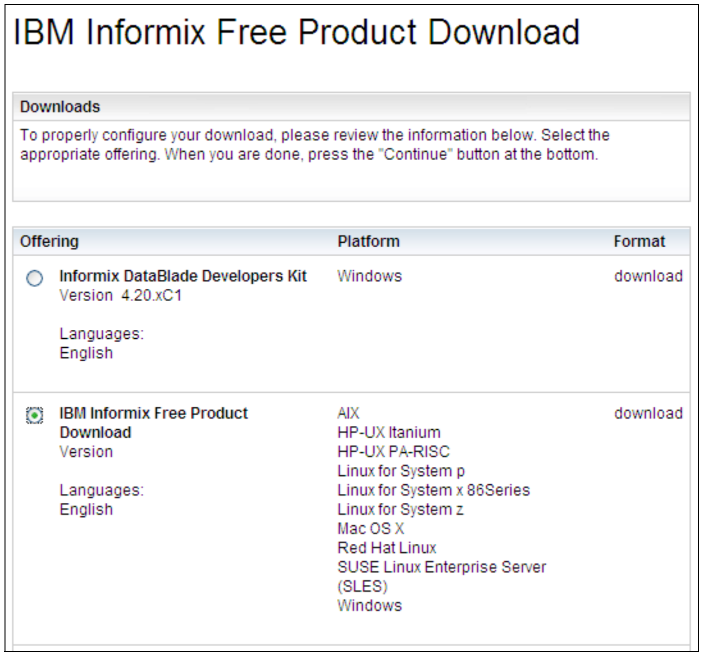
2.2 Downloading the OAT software

To learn more about the OAT product, get the newest product information, and download the OAT product itself, go to the following address:

`http://www.openadmintool.org/`

Complete the following steps to download the OAT:

1. Registration and sign-in. You must register with the IBM Informix Free Product Download website to download Informix at no cost or trial software products, including OAT. If you have already registered, sign in using your registered IBM ID. If you are not registered, click **Register Now** and complete the registration form.
2. After login, select **IBM Informix Free Product Download** in the IBM Informix Free Product Download window (Figure 2-1).



IBM Informix Free Product Download

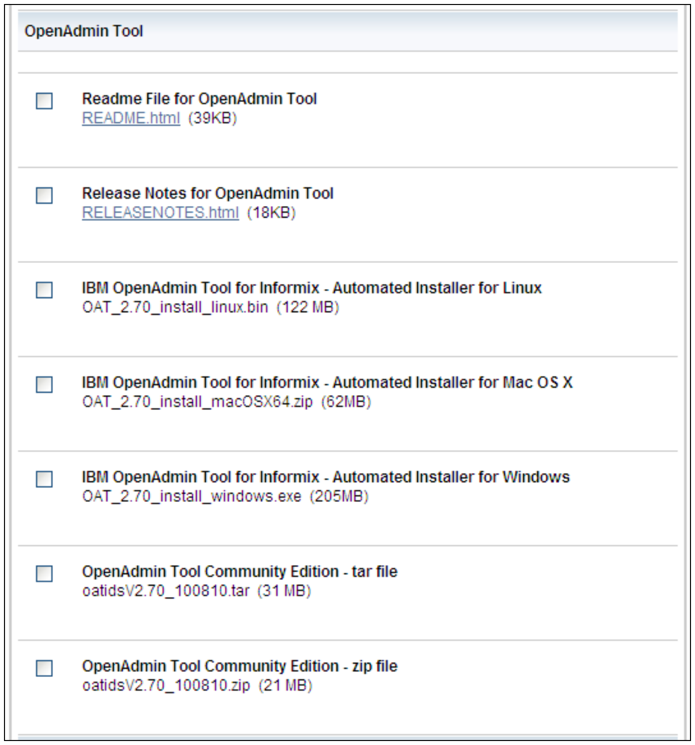
Downloads

To properly configure your download, please review the information below. Select the appropriate offering. When you are done, press the "Continue" button at the bottom.

	Offering	Platform	Format
○	**Informix DataBlade Developers Kit** Version 4.20.xC1 Languages: English	Windows	download
●	**IBM Informix Free Product Download** Version Languages: English	AIX, HP-UX Itanium, HP-UX PA-RISC, Linux for System p, Linux for System x 86Series, Linux for System z, Mac OS X, Red Hat Linux, SUSE Linux Enterprise Server (SLES), Windows	download

Figure 2-1 Choose IBM Informix Free Product Download to continue

3. In the product list window, scroll down to the OpenAdmin tool section and select the OAT package for your platform (Figure 2-2). Click **Download Now** at the bottom of the page. A Java based application called Download Director starts and downloads the distribution file to the local disk.



OpenAdmin Tool

☐ Readme File for OpenAdmin Tool
README.html (39KB)

☐ Release Notes for OpenAdmin Tool
RELEASENOTES.html (18KB)

☐ IBM OpenAdmin Tool for Informix - Automated Installer for Linux
OAT_2.70_install_linux.bin (122 MB)

☐ IBM OpenAdmin Tool for Informix - Automated Installer for Mac OS X
OAT_2.70_install_macOSX64.zip (62MB)

☐ IBM OpenAdmin Tool for Informix - Automated Installer for Windows
OAT_2.70_install_windows.exe (205MB)

☐ OpenAdmin Tool Community Edition - tar file
oatidsV2.70_100810.tar (31 MB)

☐ OpenAdmin Tool Community Edition - zip file
oatidsV2.70_100810.zip (21 MB)

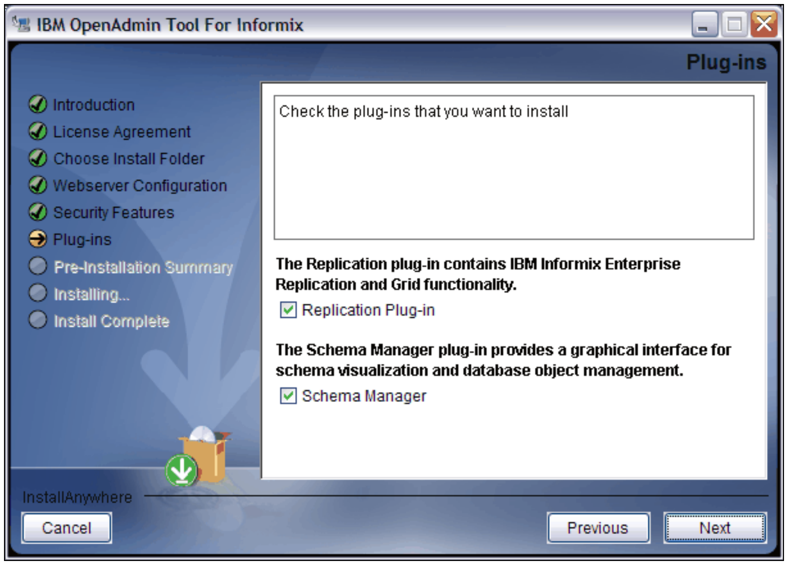
Figure 2-2 OAT download offerings

2.3 Installing OAT

After you successfully downloaded the OAT files, you can continue with the installation of the tool. The target installation machine for the OAT does not need to have a database server installed, but should be able to connect to the host machine where the database server resides. You need to have a web browser installed on the machine to execute OAT.

2.3.1 Installation

The installation process is straightforward. The OAT installation file downloaded is an executable. Run the file and complete with the steps of the installation routine. The essential point in the installation for managing Informix clusters, such as MACH11, Enterprise Replication, and Flexible Grid environments within the OAT, is selecting the Replication Plug-in. Make sure both the Replicating Plug-in and Schema Manager are selected in the Plug-Ins step (Figure 2-3).



Check the plug-ins that you want to install

The Replication plug-in contains IBM Informix Enterprise Replication and Grid functionality.

- [✓] Replication Plug-in

The Schema Manager plug-in provides a graphical interface for schema visualization and database object management.

- [✓] Schema Manager

Figure 2-3 Replication Plug-in selection

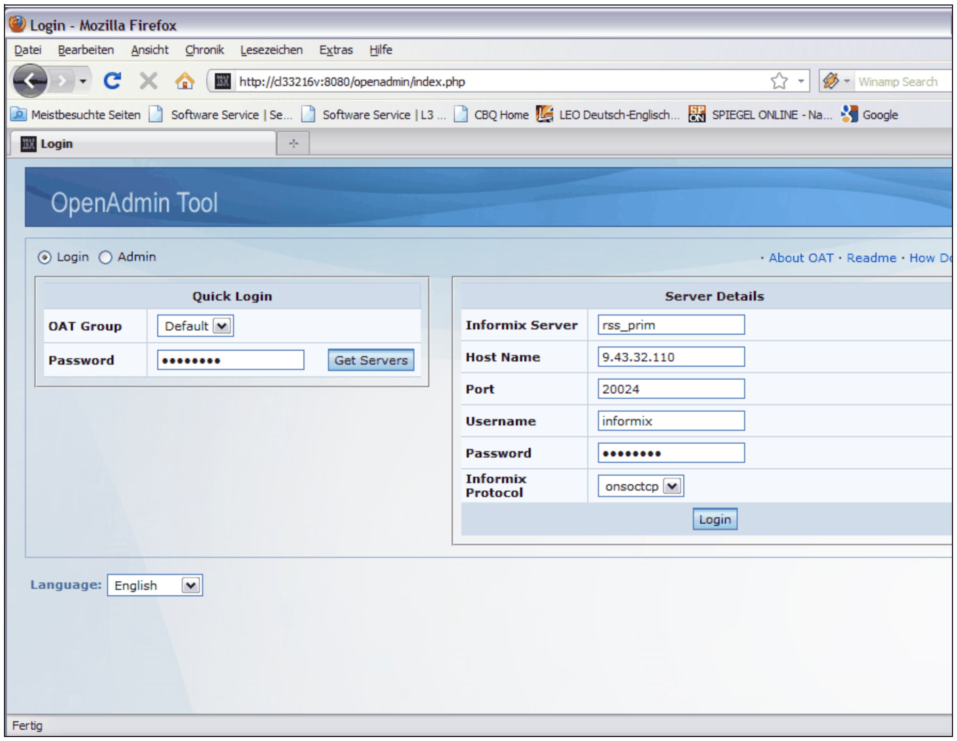
After you complete all the required definitions for the installation, such as the installation directory, plug-in selection, and administration password, the installation routine installs the product.

First startup

After a successful installation, check the status of the OAT by starting it through a browser on the installation machine using the following address:

```
http://localhost:8080/openadmin/index.php
```

The specified port for the web server might vary depending on your configuration settings. You should see the following entry point for the OAT in your browser (Figure 2-4).


Login - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://d33216v:8080/openadmin/index.php
Meistbesuchte Seiten Software Service | Se... Software Service | L3 ... CBQ Home LEO Deutsch-Englisch... SPIEGEL ONLINE - Na... Google
Login
OpenAdmin Tool
Login Admin
About OAT · Readme · How D
Quick Login
Get Servers
Server Details
Login
Language: English
Fertig

Quick Login	
OAT Group	Default
Password	••••••••

Server Details	
Informix Server	rss_prim
Host Name	9.43.32.110
Port	20024
Username	informix
Password	••••••••
Informix Protocol	onsoctcp

Figure 2-4 Entry point for starting the OAT

Manual installation adjustments

The OAT installation package is a bundle and contains the following products:

- Apache web server
- PHP environment embedded in Apache
- IBM Informix Connect
- The OAT application itself

All the packages are automatically installed on the target machine. If you have some of the software already installed, you can remove or deactivate the redundant software afterwards.

Apply the following changes to the `httpd.conf` configuration file on the Apache web server:

- Add the INFORMIXDIR environment variable.
- Add the location of the PHP shared library installed by the OAT package.
- Apply the path of your `php.ini` file.

Here is an example of the new configuration parameters added to the `http.conf` file on Windows:

```
LoadModule php5_module "C:\Program Files\OpenAdmin\PHP_5.2.4\php5apache2_2.dll"
AddType application/x-httpd-php .php
PhpIniDir 'C:\Program Files\OpenAdmin\PHP_5.2.4'
setenv INFORMIXDIR 'C:\Program Files\OpenAdmin\Connect_3.70'
```

On the PHP side, if you already have a PHP environment in place, you must include the extension entries in the `php.ini` file and copy the appropriate extension libraries into your installation directory. Most important is to include the Informix_PDO extension to your existing installation to ensure a successful connection to the remote database server. Here is an example of the Informix_PDO extension entry added to the `php.ini` file:

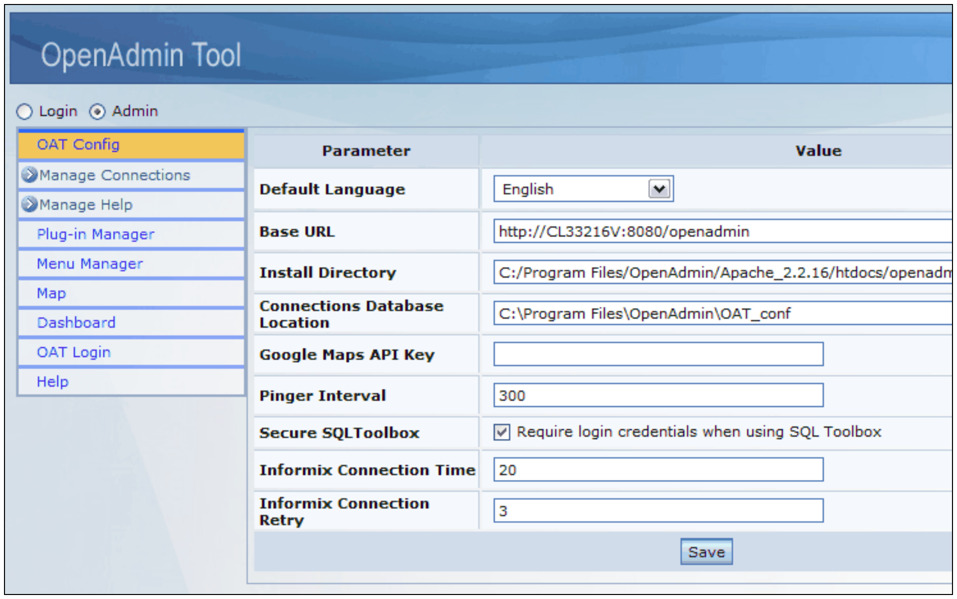
```
extension_dir = C:\Program Files\OpenAdmin\PHP_5.2.4\ext
extension=php_pdo_informix.dll
```

OAT uses SOAP. If your system has SOAP already, include it in the `php.ini` file. If you do not have that file, use the one included in the shipped PHP executable.

After the adjustments are complete, from the OAT entry page, you should be able to either connect to an Informix instance by specifying the required connection parameters or open the administration page for the OAT.

2.3.2 Configuration

The installation routine sets certain OAT tool parameters during installation. You can verify and change these predefined settings by selecting **Admin** → **OAT Config**. Figure 2-5 shows the administration page. Adjust the settings to your needs.



Parameter	Value
Default Language	English
Base URL	http://CL33216V:8080/openadmin
Install Directory	C:/Program Files/OpenAdmin/Apache_2.2.16/htdocs/openadm
Connections Database Location	C:\Program Files\OpenAdmin\OAT_conf
Google Maps API Key	
Pinger Interval	300
Secure SQLToolbox	✓ Require login credentials when using SQL Toolbox
Informix Connection Time	20
Informix Connection Retry	3

Figure 2-5 Administration window of the OAT

To see the coordinates of the location of your servers, you need a Google Maps API key, which can be acquired from the following address:

`http://www.google.com/apis/maps/signup.html`

Enter the key into the Google Maps API Key field. You can find, for example, the specific latitude and longitude of a server.

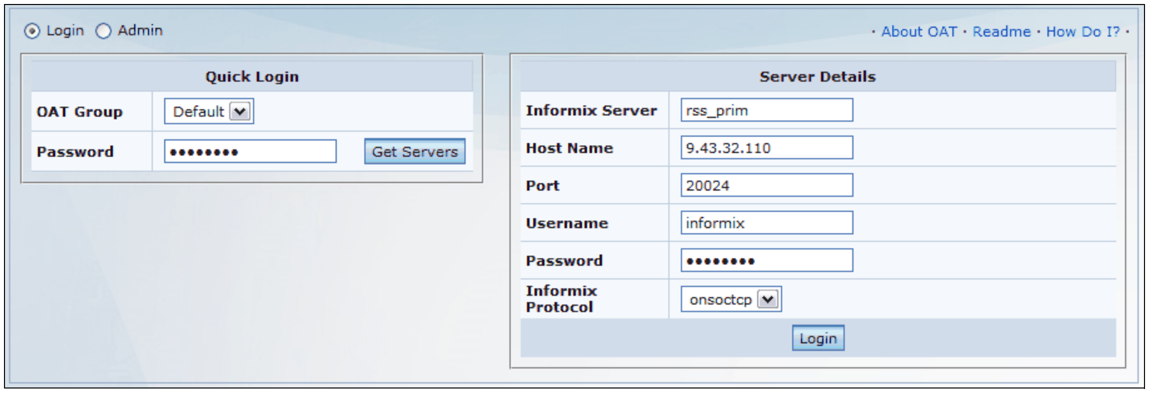
2.4 Database administration and monitoring with OAT

The major function of OAT is the administration and monitoring of the remote databases. There are extensive administration functionalities provided by the tool. Here we provide a high level overview of the OAT functions, focusing on the maintenance of clustered environments.

2.4.1 Accessing a remote database server

To access the Informix database server, specify the connection parameters in the Sever Details section of the OAT entry web page (Figure 2-6):

- Database server name: Refer to the `onconfig` file of the server configuration.
- IP address of the host machine of the database server: Refer to the `sqlhosts` file of your target database server configuration.
- Port number used by the database server.
- User and password of a trusted user on this host.



Quick Login

OAT Group	Default
Password	••••••••

Server Details

Informix Server	rss_prim
Host Name	9.43.32.110
Port	20024
Username	informix
Password	••••••••
Informix Protocol	onsoctcp

Figure 2-6 Server details

After establishing a successful connection, you see the administration console for the specific database server (Figure 2-7).

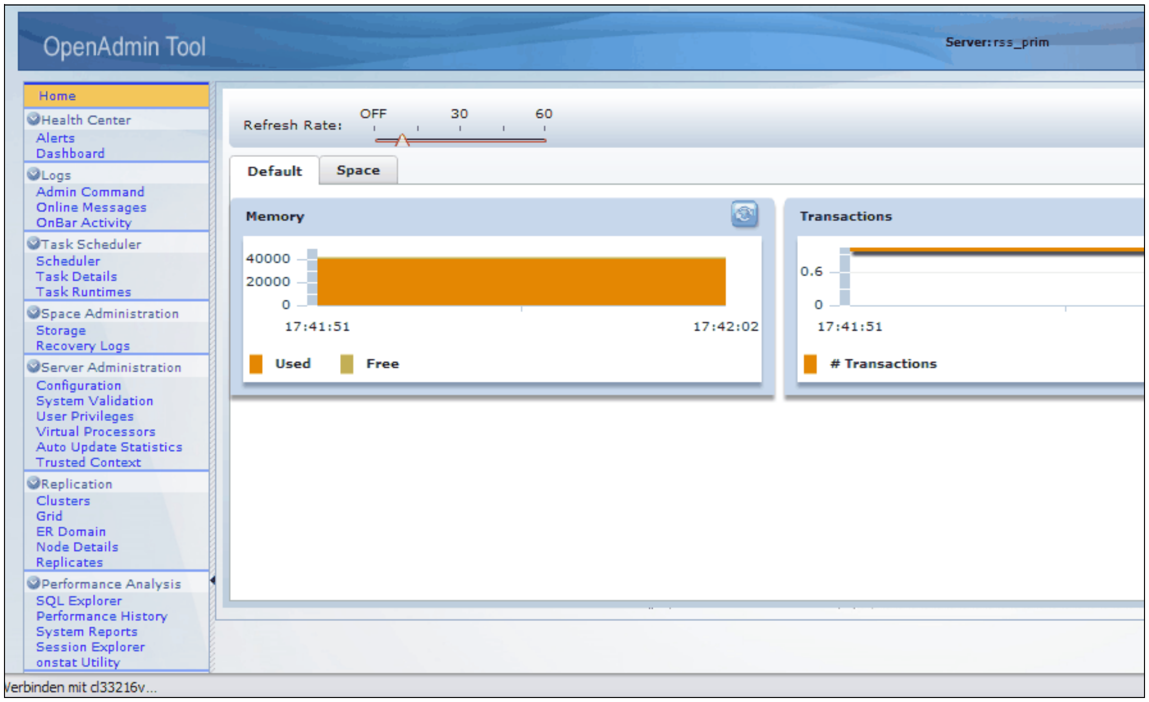


Figure 2-7 Entry administration point for an Informix database server

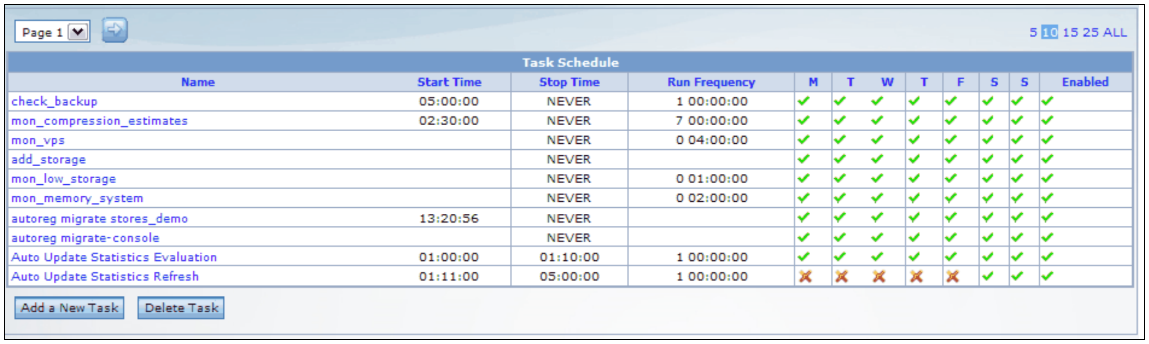
2.4.2 OAT core functionality overview

OAT provides the following core functionalities for managing and monitoring the database server:

- Monitoring
  - Monitoring database server alerts
  - Monitoring the database server log files, such as the `online.log` and the backup log

– Running onstats remotely

– Monitoring and maintaining the task scheduler (Figure 2-8)


Page 1

5 10 15 25 ALL

Task Schedule

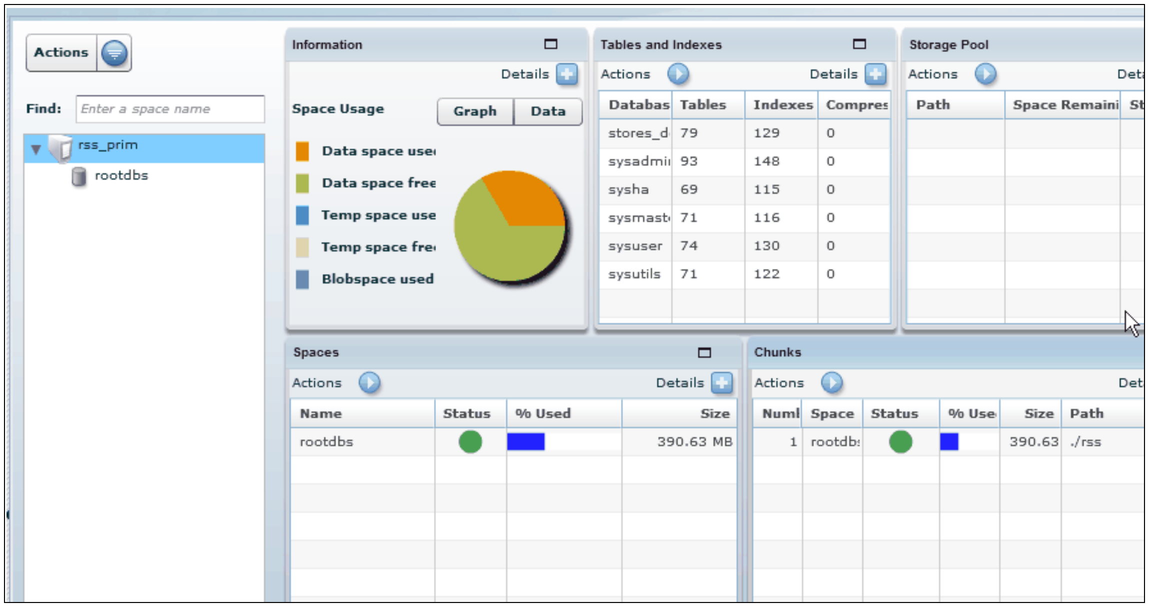
Name	Start Time	Stop Time	Run Frequency	M	T	W	T	F	S	S	Enabled
check_backup	05:00:00	NEVER	1 00:00:00	✓	✓	✓	✓	✓	✓	✓	✓
mon_compression_estimates	02:30:00	NEVER	7 00:00:00	✓	✓	✓	✓	✓	✓	✓	✓
mon_vps		NEVER	0 04:00:00	✓	✓	✓	✓	✓	✓	✓	✓
add_storage		NEVER		✓	✓	✓	✓	✓	✓	✓	✓
mon_low_storage		NEVER	0 01:00:00	✓	✓	✓	✓	✓	✓	✓	✓
mon_memory_system		NEVER	0 02:00:00	✓	✓	✓	✓	✓	✓	✓	✓
autoreg migrate stores_demo	13:20:56	NEVER		✓	✓	✓	✓	✓	✓	✓	✓
autoreg migrate-console		NEVER		✓	✓	✓	✓	✓	✓	✓	✓
Auto Update Statistics Evaluation	01:00:00	01:10:00	1 00:00:00	✓	✓	✓	✓	✓	✓	✓	✓
Auto Update Statistics Refresh	01:11:00	05:00:00	1 00:00:00	✗	✗	✗	✗	✗	✓	✓	✓

Add a New Task | Delete Task

Figure 2-8 Task scheduler overview

► Resource management

– Monitoring and maintaining the database storage and the physical and logical log (Figure 2-9)



Databas	Tables	Indexes	Compres
stores_d	79	129	0
sysadmi	93	148	0
sysha	69	115	0
sysmast	71	116	0
sysuser	74	130	0
sysutils	71	122	0

Path	Space Remaini	St

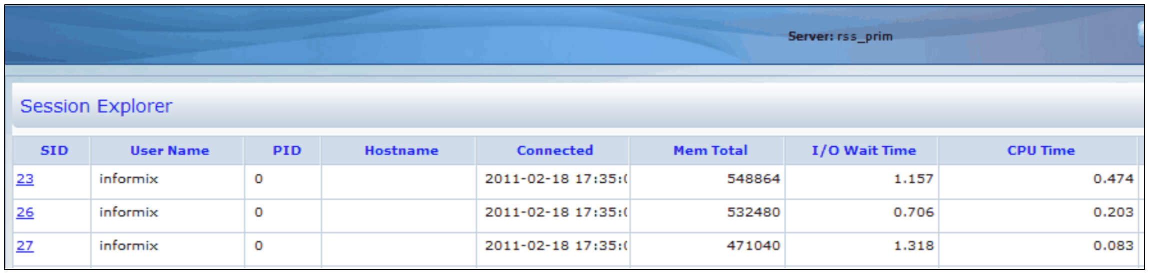
Name	Status	% Used	Size
rootdbs	●		390.63 MB

Numl	Space	Status	% Use	Size	Path
1	rootdb:	●		390.63	./rss

Figure 2-9 Maintaining disk space in OAT

– Dynamic maintenance of virtual processor resources, such as CPU and AIO virtual processors

– Session based resource monitoring for I/O, memory, and CPU utilization (Figure 2-10)

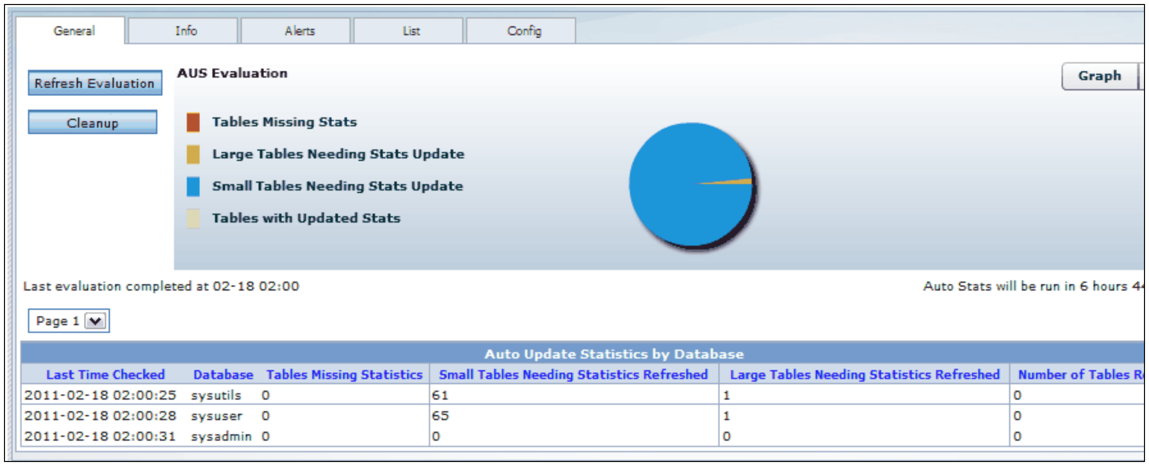

Server: rss_prim

Session Explorer

SID	User Name	PID	Hostname	Connected	Mem Total	I/O Wait Time	CPU Time
23	informix	0		2011-02-18 17:35:	548864	1.157	0.474
26	informix	0		2011-02-18 17:35:	532480	0.706	0.203
27	informix	0		2011-02-18 17:35:	471040	1.318	0.083

Figure 2-10 Session monitoring in terms of resource utilization

- Validating and adjusting database server configuration parameters on demand
- Consistency checking on the data in your database server
- Cluster management for Flexible Grid, MACH11, and ER clusters
- Database schema visualization and maintenance
- Performance:
  – Performance monitoring based on Query analysis
  – Automatic update statistics fertilization and adjustment (Figure 2-11)



AUS Evaluation

🟥 Tables Missing Stats
🟨 Large Tables Needing Stats Update
🟦 Small Tables Needing Stats Update
⬜ Tables with Updated Stats

Last evaluation completed at 02-18 02:00

Auto Stats will be run in 6 hours 4

Auto Update Statistics by Database

Last Time Checked	Database	Tables Missing Statistics	Small Tables Needing Statistics Refreshed	Large Tables Needing Statistics Refreshed	Number of Tables R
2011-02-18 02:00:25	sysutils	0	61	1	0
2011-02-18 02:00:28	sysuser	0	65	1	0
2011-02-18 02:00:31	sysadmin	0	0	0	0

Figure 2-11 Automatic update statistics maintenance

3

Configuring a server for Flexible Grid and Enterprise Replication environments

A database server replication network represents a reflection of a company’s business infrastructure and captures the communication between various organizations. It is the basis for their data exchange. The replication topologies can vary in a wide range from fully connected participants exchanging data with others to a hierarchical one-to-many data flow direction.

Designing an suitable replication topology is the foundation for establishing successful IBM Informix Flexible Grid and Enterprise Replication cluster environments. Therefore, capturing the requirements for the communication flow of the replicated data is an essential task.

Planning and realizing a database server network that matches the data flow in your organization is the intention of this chapter. We introduce the common use cases of replication environments for your reference when you are selecting the replication infrastructure. We discuss the configuration and setup tasks for defining the replication server.

3.1 Basic concepts

Informix provides a variety of cluster technologies that offer the capabilities to have one-to-many database servers sharing the same disk or to have their own copy of the data in ready-only read-write fashion. This combination of technologies offers you a wide spectrum of solutions for data availability and workload distribution.

So, why do you need additional replication?

The answer is flexibility. The Informix replication feature gives you flexibility in the database version and operating system platform. In an Informix replication environment, each of the database servers can have different Informix versions. You can replicate data from a Windows system to an UNIX system and across various UNIX operating systems. You can define which data must replicate and in which direction. You can define the replication hierarchies. The replication environments are much more flexible in supporting the data flow required for your business. All replication servers are fully operational and using the Flexible Grid that allows you to replicate the database schema changes in addition to the data changes across the replication network.

Replication environments can be combined with a MACH11 cluster and can be included in smart client distribution solutions to use all available resources and to separate clients according their resource needs.

The setup of an database server replication network consist of two parts:

- Logical design: The logical design is the conceptual parts that lay down the foundation. The tasks include:
  - Identifying the organizations in the enterprise that can benefit from replication.
  - Gathering the replication data flow requirements and identifying the flow constraints and hierarchies.
  - Designing the replication topology that meets the business needs.
- Physical implementation: This part includes:
  - Identifying the existing resources, including a database server, host machines, and storage systems.
  - Ensuring that the resources (CPU, I/O, and network resources) have enough bandwidth for the additional workload requirements.
  - Integrating the new replication network.

3.1.1 Business models applicable for replication

An enterprise typically has a number of different requirements for using data replication. One key to designing a database server replication network is a clear understanding of those requirements. Some organizations might want to increase fault tolerance, thereby duplicating the critical information in more than one location, or some might want to increase data availability and performance of their application by using local replicates to avoid WAN network traffic.

Here are some examples of the business requirements for which organizations might want to replicate data:

- Call centers: Organizations may want to have a consistent set of records that can be updated at any site in a peer-to-peer fashion. For example, an international corporation might want to be able to support operations 24x7 and choose to have the support centers in various locations around the world. When a call is made at midnight in Dallas, it might be routed to Mumbai, India. The information taken would then be replicated to other call centers in locations such as London and Denver. That way, each call center would have information that was captured at all other call centers.
- Data distribution: This is the replication of data from a single source to multiple targets. An example is replicating prices or inventory from a corporate headquarters to each remote location. The data from the single central database is distributed to the many location, which is a one-to-many way transfer. The Flexible Grid advances the concept of only replicating the data with the ability of also replicating schema changes.
- Workflow management: In this case, the different parts of a workflow are processed on separate systems. The results of the work on each system are replicated to the system on which the next step will be performed. An example might be insurance claims processing. The adjuster might enter the damage claim on one system, and it may be reviewed or adjudicated by a reviewer on a different system using a separate database. In this case, data is moved from one system to another among a set of peer database severs. It is not necessarily a one-way transfer, but that is often the case.
- Data warehouse: This is the primary repository of an organization for historical data. Information is collected at one central location from multiple sources, and made available to all locations. The critical factor leading to the use of a data warehouse is that a data analyst can perform complex queries and analysis, such as data mining, on the information without impacting the operational systems. Collected data can also be used for management decision support systems.

Based on these business models, we can categorize the type of replication as either primary-target or update-anywhere. In a *primary-target* replication, data is replicated from one system to one or more other systems, often for the purpose of data consolidation or distribution. In an *update-anywhere* replication system, data is replicated among a set of database servers where any of them may be updated. This type of replication is often used for workload sharing.

3.1.2 Common replication topologies

Given the purposes, types, and use cases for a required replication scheme, a topology can be designed. There are two major types of topologies commonly used. They are:

- Fully-connected database servers (update-anywhere)
- One-to-many topology (primary-target between multiple servers)

The benefit using these kind of topologies is their simplicity of administration such as setup, monitoring, and data flow tracking. When planning the layout of a new database server network for replication, keep the flow simple and try to avoid complexity.

Of course, there are other possible topology setups that can be used as the basis for Flexible Grid and ER environments. A hierarchical tree of database servers defining additional communication layers in primary-target or update-anywhere servers could be a possible infrastructure. A group of database servers would advance the topology of the hierarchical tree.

3.1.3 Fully-connected database servers

In a fully-connected topology, each database server has a connection to each other database server (Figure 3-1). Any changes of the data can be applied locally and transferred directly to any database server or a subset of the others.

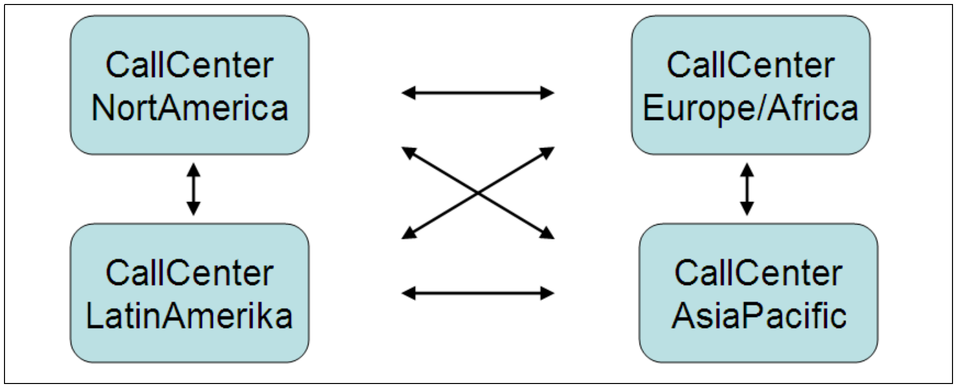


Figure 3-1 Fully connected database server within a call center organization

This kind of topology can be applied to an existing call center organization. You have problem tracking around the clock for clients operating globally with the opportunity to shift the ownership of open issues across different time zones. In this type of database server infrastructure, all the data needs to be available and maintainable at all participating locations.

An advantage of this topology is that data can be transferred directly from any machine to any of the others without needing an additional system in between to re-broadcast the replication messages. The disadvantage of this topology is that the number of connections from one database to other grows quickly with the addition of new nodes, so it is extremely impractical for large number of replication servers in a domain. However, the topology is generally used when there are only a small number of nodes to be interconnected.

Another thing to consider is database security. Do you want every database server in replication domain connected to all other database servers? if not, then the fully connected topology may not be the best option.

3.1.4 One-to-many database server organized in a star topology

In regards to data flow, one-to-many organized database servers are commonly used in retail businesses. You may define one master database instance and this information is shared across its subsidiaries. Any changes applied either to the schema (when using a Flexible Grid) or data is executed only once, initiated on one machine, and replicated and applied to the other machines. When planning this kind of infrastructure, the master database server becomes a highly critical resource. This infrastructure definitely requires additional resources, such as an RSS or HDR cluster, to ensure the availability of the data. When using a cluster environment for your master database server, only the primary database server of the cluster participates in the replication. An high-availability (HA) cluster has no greater capacity for handling the replication or the Flexible Grid than a single database server. The replication considers the high-availability cluster as a single logical instance. In the event of a primary server failure, the secondary server can be promoted to primary or standard server, and data replication resumes.

Another possible scenario is an organization running multiple remote locations that needs to provide general information to these locations, such as travel guidelines, HR information, and management directives. This information is maintained on a central server and is replicated to local systems.

Figure 3-2 shows a one-to-many database base infrastructure.

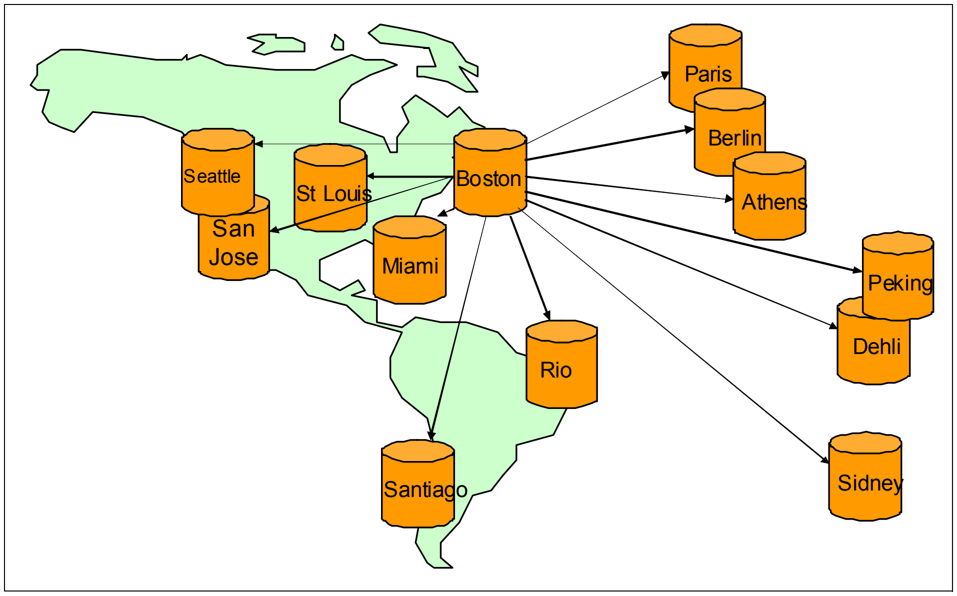


Figure 3-2 One-to-many database server replication topology

3.1.5 Mixed topology

Depending on your requirements, you may want to use either a fully connected or hierarchical topology, or use both together. When both topologies are used together, we refer to it as a *forest of trees* topology. This topology is a set of hierarchical trees in which the root nodes of the trees are fully connected. An example of this topology is shown in Figure 3-3.

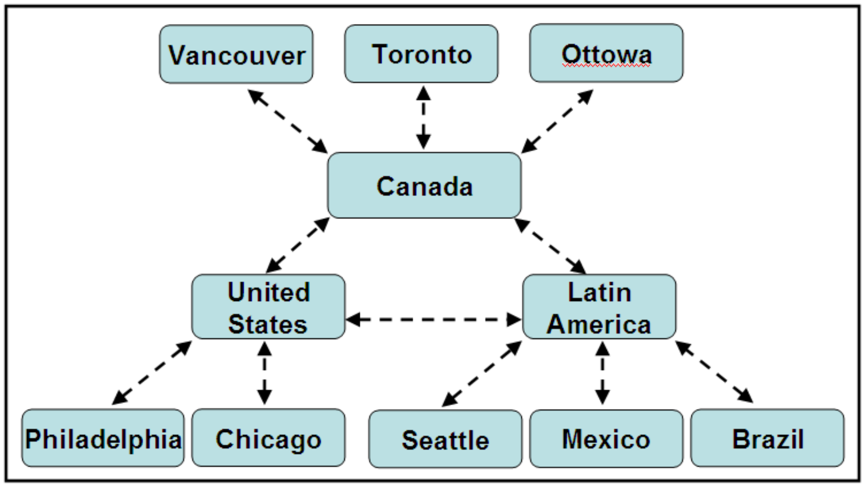


Figure 3-3 A mix of fully connected and hierarchy topology

In this example, the US, Canada, and Latin America servers are peers, and each one is also the root of a tree. There is no requirement for the trees to be balanced or to have the same number of levels. What is important is that the root or hub nodes are highly available to have uninterrupted replication among child nodes.

3.2 Building a replication environment

After a topology has been chosen, you have finished the conceptual part of the infrastructure setup. You have defined the directions of the communication and the data flow. You can start constructing the replication domain.

The setup steps for Flexible Grid and ER are slightly different. The steps are as follows:

- General setup requirements:
  a. Configure the database server to ensure that the resources for replication are available. Replication requires additional dbspace resources and instance configuration changes.
  b. Make sure that the communication between the database server is correctly configured. A trusted connection between each pair of nodes that must pass replication data is required. Some additional entries in the `sqlhost` file, such as server groups, have to be applied.
  c. Define the replication servers. The local database server is attached to the network and the communication flow to other server is defined.
- Continuing steps for Flexible Grid setup
  a. Define the grid, including the participating database server, permissions, and a master database server.
  b. Enable the grid.
- Continuing steps for an ER environment setup
  a. Prepare the databases for replication by adding the shadow columns where they are required. These added columns may change the number of rows per page and the amount of disk space required for the database(s).
  b. Produce (design, write, test, and install) any stored procedures that may be necessary for conflict resolution.
  c. Define the replicates and replicate sets. This specifies what data (tables or parts of tables) that are to be replicated as well as the source and target of each replicate.
  d. Start the replicates or replicate sets. This step begins the actual transfer of data. Up to this step, the work has been to configure and identify things. This final step begins moving data.

In the following sections of this chapter, we focus on the description of the setup of the underlying database server replication and grid network. This includes the specification of the required changes in the database server configuration files as well as why to use additional spaces and their relationship to the replication configuration parameter. We provide best practices guidance. The major focus here is on which replication configuration parameter affects the performance and how it can be improved. We also show how to attach the database server to a new or existing replication network.

We discuss more details about the Flexible Grid setup and the SQL interface in Chapter 4, “IBM Informix Flexible Grid” on page 57. In Chapter 5, “Administering Enterprise Replication” on page 101, we provide detailed discussion for setting up an operational Enterprise Replication environment.

3.3 Defining a sample replication infrastructure

We start our discussion by introducing a sample environment that we use to illustrate the definitions and settings that ER provides with regard to the replication server and replicates. We provide detailed examples based of those definitions and settings.

Consider a company operating in a global environment that has subsidiaries on all continents. For delivery, reporting, sales, and revenue accounting, it is necessary that the database servers situated on each continent are linked together and can exchange data. For the business in Europe, there is a additional requirement of a master database server that maintains the data replication to the local hubs. The company has defined the server *newyork* as the database structure owner. Figure 3-4 shows the topology.

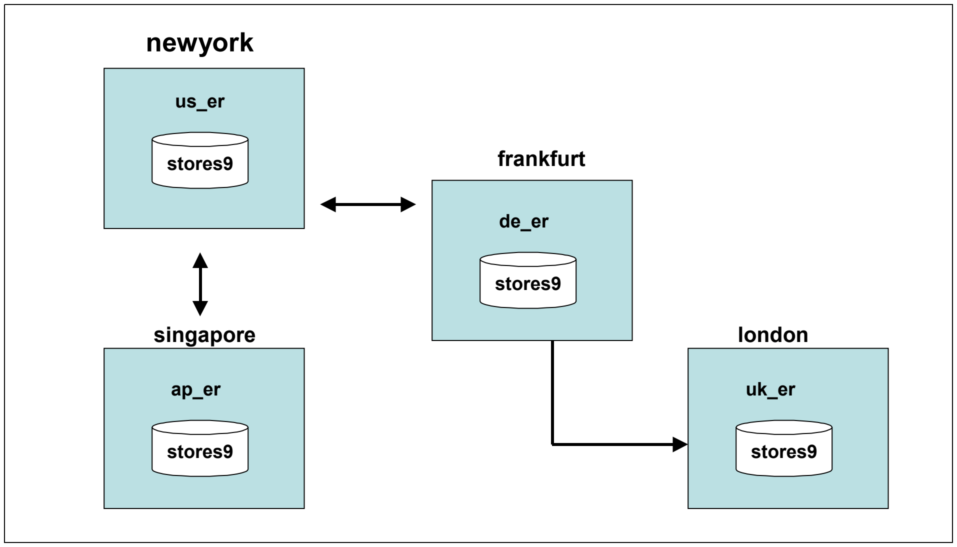


Figure 3-4 Sample replication topology using different replication server types

The plan is to use four replication servers in the sample environment. The newyork server is defined as the master, or root replication server. The *london* and *singapore* servers are defined as leaf servers and *frankfurt* is a non-root server. In the hierarchy, frankfurt and singapore are directly connected to newyork. The london hub has an additional ER leaf node that is connected to frankfurt. In the communication flow, newyork, frankfurt, and singapore act in an update-anywhere environment. Data changes between frankfurt and london are replicated in a primary-target fashion.

3.4 Preparing the database server

Preparing the database server and attaching the server to the replication network includes the following major steps:

- Ensure data consistency across the data set that must be replicated.
- Modify the configuration file.
- Review the size and types of the existing dbspaces.
- Attach the database server to the replication network.

3.4.1 Data consistency across database servers in the replication network

Before you can start any configuration changes on the local database server, you must check the consistency for all the data. Focus especially on the data that will be affected by the replication to avoid collisions and unexpected errors. Check the content and the schema of the base tables for any differences. There are two methods for ensuring consistency:

- Manual check
  - Define one of the server in the replication network as the master of the data and the database schema.
  - Check the schema of the base tables, compare the data, identify the delta data, load the missing data, and update the table schema based on guidelines from the master server.
- Clone the new database server
  - Use the ifxclone or restore utilities to set up a new server to make sure that the new server meets the consistency requirements. We provide details about using ifxclone to create new grid and ER servers in Chapter 14, “Using ifxclone” on page 525.

3.4.2 Adding server groups to the sqlhosts file

After you ensure data consistency, enable communication between the servers participating in the replication network or grid. To define a new replication server, specify a unique group name and a group ID as a separate entry in the `sqlhosts` file for each server in the ER or Flexible Grid topology. If the database server is already in the `sqlhost` file, append `g=<id>` to the entry.

Make sure that the systems for the database server are trusted for the informix user on each communication pair. Example 3-1 shows the `sqlhosts` file for our sample topology.

Example 3-1 The sqlhosts file using server groups for the sample replication network

```
newyork group          -                       - i=2930
us_er onsoctcp       100.1.1.1 23467 g=newyork

frankfurt group          -                       - i=2932
de_er onsoctcp       100.2.1.1 23465 g=frankfurt

london group          -                       - i=2934
uk_er onsoctcp       100.2.2.1 23468 g=london

singapore group          -                       - i=2936
ap_er onsoctcp       100.3.1.1 23469 g=singapore
```

3.4.3 Defining ER related dbspaces

The complete ER functionality relies on a system defined and system created database named *syscdr*. This database is created automatically on each server when the server is defined as a replication server. Before the transition to a replication server is made, you must specify a normal dbspace for the location of the syscdr database. In addition, a dbspace for the send and receive queue header data and an sblob space for the data that will be sent out for application to the target replication server is needed. The dbspace location of the syscdr and the queue header information can be the same.

Check the existing dbspaces and chunks for their utilization. Add additional resources if required. Create at least one sbspace in the system if currently no sbspace exists. Using the OAT later on for the replication server definition, you do not need to create the dbspaces locally on the database server at this time; you can create them from the browser.

3.4.4 The onconfig parameter and replication environments

There are several parameters in the `onconfig` files which may be set for the setup of the replication. They maintain database server resources, such as disk space, memory, and amount of threads granted to the replication subsystem. There are the following replication specific parameters:

- CDR_DBSPACE

  Defines the location of the syscdr database at replication server definition time. The values defines the name of a normal existing dbspace. The default is the rootdbs.

- CDR_QHDR_DBSPACE

  Defines the location of the internal maintenance information for the spooled transaction data that has to be replicated to one or more targets. The value specifies a normal existing dbspace. The default is the same name as specified in CDR_DBSPACE.

- CDR_QDATA_SBSPACE

  Defines the location of the spooled transaction information. This is stored in a sblob. The value specifies a sblob space that has to exist at the definition of the replication server. There is no default.

- CDR_MAX_DYNAMIC_LOGS

  Specifies the number of dynamic log allocation requests that can be issued by ER components to prevent blocking the system with long transactions. The default value is 0.

- CDR_EVALTHREADS

  - Defines the number of grouper threads needed for transaction activity extraction, determines which parts of the transaction are covered by defined replicates, compresses activities on the same row, and prepares the transactions for sending.
  - Specifies two comma separated values where the first value defines the number of grouper threads per CPU VP and the second one specifies the number of additional threads independent of the number of CPU VPs. The default setting is 1,2, which means for a one CPU configured system, the total number of grouper threads is 3.

- CDR_DSLOCKWAIT

  ER supports, on the targets, a parallel application of the incoming transactions. In addition, there can be lock wait situations between the DS threads and the normal workload. This parameter defines the lock wait value for the Informix threads. The default is 5.

- CDR_QUEUEMEM

  Defines the amount of memory for the internal memory queues used by the grouper threads and for maintaining the transactions to be sent to the targets. The default value is 4096 KB.

- CDR_NIFCOMPRESS

  This parameter allows specification of a certain compression level to reduce the volume of data sent over the network initiated by the Network InterFace (NIF) threads. The default value is 0.

- CDR_SERIAL
  - CDR_SERIAL allows the DBA to define a unique range of values for each replication server in the ER infrastructure to avoid replication conflicts with serial columns.
  - A specific delta value for the increment and an offset for the start. We suggest starting all the servers with a offset in the same range, but with a different delta to achieve uniqueness.

Change the parameter to the appropriate settings. Most of them can be modified in the running system using the cdr utility without restarting the instance if the current setting need to be changed.

3.5 Defining the replication server

The last task of server preparation for setting up a Flexible Grid or ER environment is to attach the local server to a new or existing Flexible Grid or ER database server infrastructure.

There are two different methods to define the replication server:

- Using the OAT administration console: You can set up a complete infrastructure remotely through this menu-driven tool from a single execution point.
- cdr utility: This is command-line utility for creating and running setup scripts or for embedding the creation process. You can run the cdr utility locally on the database server host system only.

3.5.1 Creating the replication server using OAT

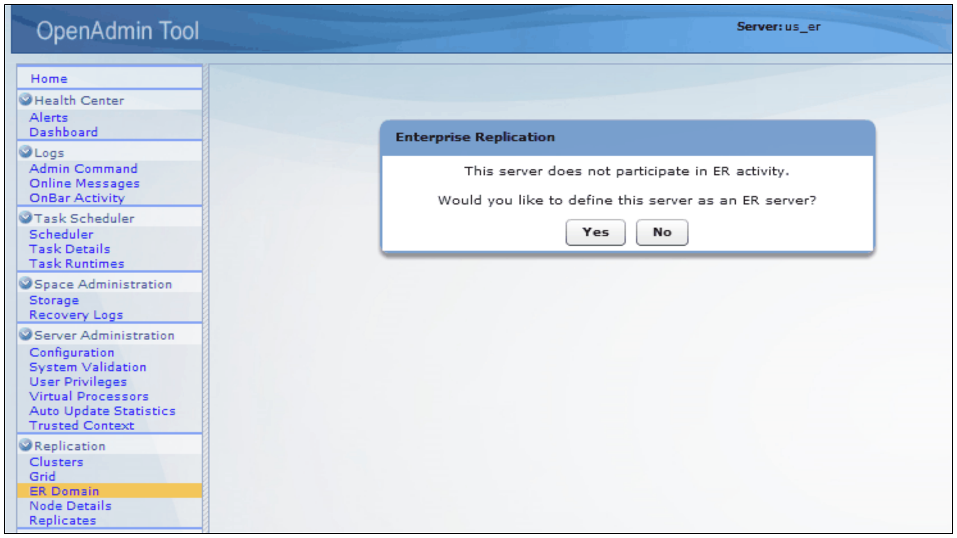
Perform the following preparation tasks for the first time using OAT to create a new replication network or to manage the existing replication network:

1. Start a browser and go to the OAT root page.
2. Select the **Admin** tab.
3. Create a new group.
4. Add all the database servers you want to connect to the replication network into the group. Specify the server by the connection parameters, such as IP address, server name, and port number.

Now you should be able to create the new infrastructure.

For a new replication server definition, continue with the following steps:

1. Select the **Login** tab in the OAT root page.
2. Specify your newly created OAT group on the left. Select the first database server you want to attach to the replication.
3. After successful connection to the server by OAT, select the ER domain provided in the Replication section on the left.
4. If the server has not been defined as a replication server, a message window opens (Figure 3-5) and prompts if you want to define the server as an ER server. Click **Yes**.

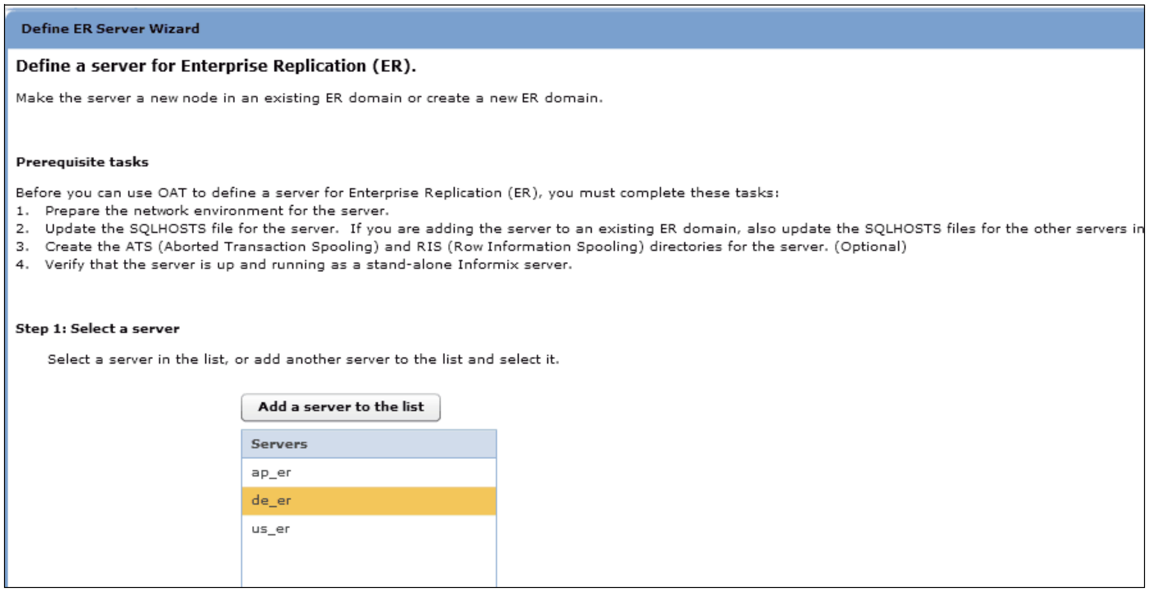


This server does not participate in ER activity.

Would you like to define this server as an ER server?

Figure 3-5 Define a new replication server startup window

5. The Define ER Server Wizard starts (Figure 3-6). Select the server you want to add to the replication from the server list. The prerequisite is that the `sqlhosts` file entries are set, the server groups defined, and the local server has been assigned to the server group.



Define ER Server Wizard

Define a server for Enterprise Replication (ER).

Make the server a new node in an existing ER domain or create a new ER domain.

Prerequisite tasks

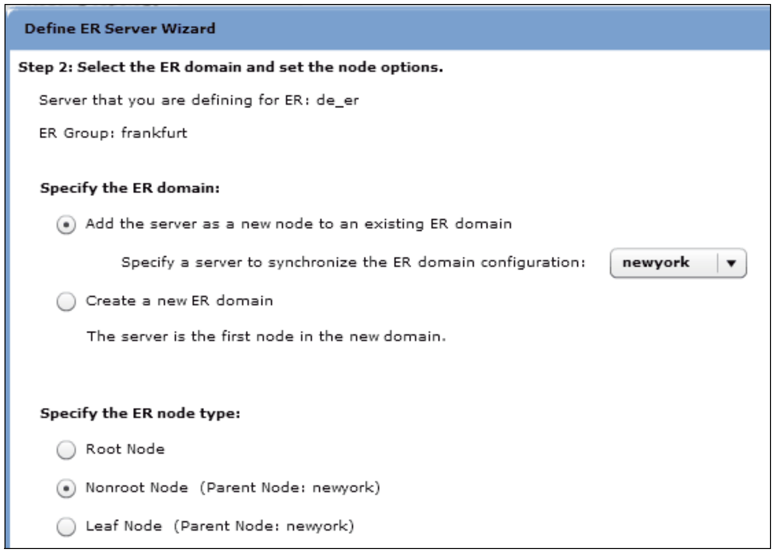
Before you can use OAT to define a server for Enterprise Replication (ER), you must complete these tasks:
1. Prepare the network environment for the server.
2. Update the SQLHOSTS file for the server. If you are adding the server to an existing ER domain, also update the SQLHOSTS files for the other servers in
3. Create the ATS (Aborted Transaction Spooling) and RIS (Row Information Spooling) directories for the server. (Optional)
4. Verify that the server is up and running as a stand-alone Informix server.

Step 1: Select a server

Select a server in the list, or add another server to the list and select it.

Figure 3-6 Select the database server that needs to be attached to the replication network

6. In the Select the ER domain and set the node options step (Figure 3-7):
   a. Specify the server type for the new replication server according to the chosen topology and the role to which the server is assigned.
   b. Specify the aborted transactions (ATS) and row information (RIS) directories to maintain possible replication conflicts and save aborted transactions or rows.



Define ER Server Wizard

Step 2: Select the ER domain and set the node options.

Server that you are defining for ER: de_er

ER Group: frankfurt

Specify the ER domain:

[✓ Add the server as a new node to an existing ER domain]

Specify a server to synchronize the ER domain configuration: newyork

Create a new ER domain

The server is the first node in the new domain.

Specify the ER node type:

Root Node

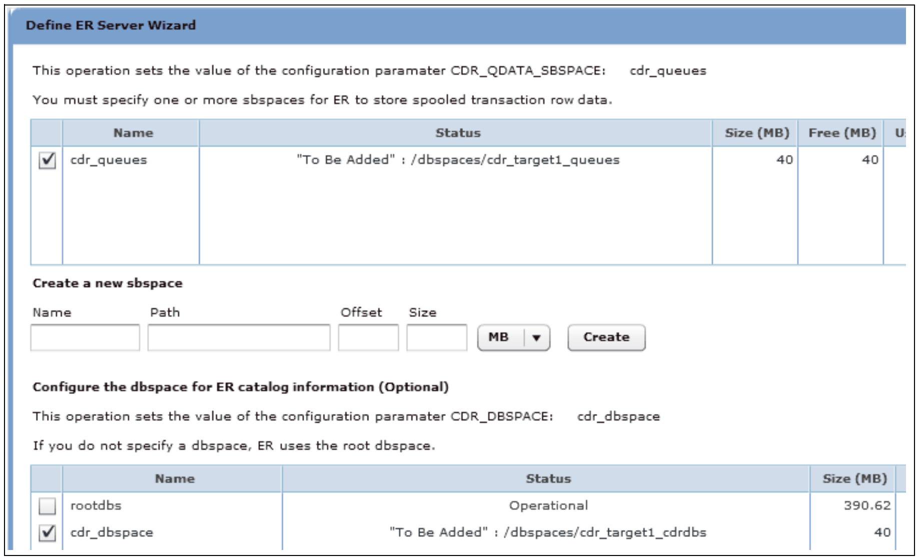
[✓ Nonroot Node (Parent Node: newyork)]

Leaf Node (Parent Node: newyork)

Figure 3-7 Define a server replication type and the ATS and RIS directories

7. The settings in this step is essential if you have not already created the dbspaces required by the replication. The dbspaces are configured by the settings of the CDR_DBSPACE, CDR_QHDR_DBSPACE, and CDR_QDATA_SBSPACE configuration parameters. Specify the name according to your settings in the `onconfig` file and define a path and a size where the initial chunks must be created.

The window for dbspace management is shown Figure 3-8.



Define ER Server Wizard

This operation sets the value of the configuration paramater CDR_QDATA_SBSPACE: cdr_queues

You must specify one or more sbspaces for ER to store spooled transaction row data.

	Name	Status	Size (MB)	Free (MB)	U
✓	cdr_queues	"To Be Added" : /dbspaces/cdr_target1_queues	40	40	

Create a new sbspace

Name	Path	Offset	Size		
				MB	Create

Configure the dbspace for ER catalog information (Optional)

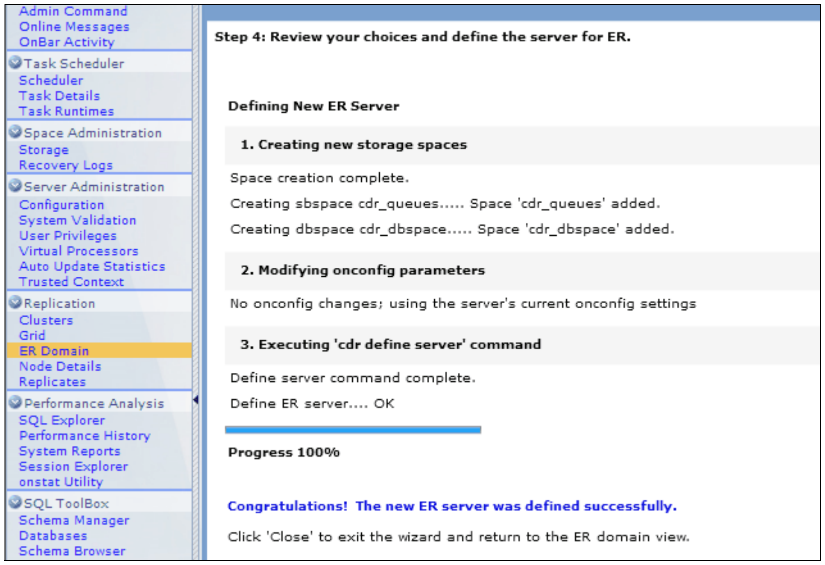
This operation sets the value of the configuration paramater CDR_DBSPACE: cdr_dbspace

If you do not specify a dbspace, ER uses the root dbspace.

	Name	Status	Size (MB)
☐	rootdbs	Operational	390.62
✓	cdr_dbspace	"To Be Added" : /dbspaces/cdr_target1_cdrdbs	40

Figure 3-8 Specify new dbspaces and chunks for the replication data

8. In the Review your choices and define the server for ER window, review the replication server definition and the execution progress and status. When the OAT defines a server successfully, a success message opens (Figure 3-9).



Step 4: Review your choices and define the server for ER.

Defining New ER Server

1. Creating new storage spaces

Space creation complete.

Creating sbspace cdr_queues..... Space 'cdr_queues' added.

Creating dbspace cdr_dbspace..... Space 'cdr_dbspace' added.

2. Modifying onconfig parameters

No onconfig changes; using the server's current onconfig settings

3. Executing 'cdr define server' command

Define server command complete.

Define ER server.... OK

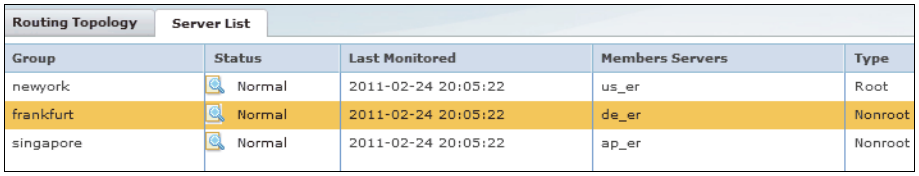
Progress 100%

Congratulations! The new ER server was defined successfully.

Click 'Close' to exit the wizard and return to the ER domain view.

Figure 3-9 Status of the execution of the replication server definition

9. Verify that the new server also appears in your topology. Our new server is added to the list of the available server in the replication network (Figure 3-10).


Routing Topology | Server List

Group	Status	Last Monitored	Members Servers	Type
newyork	Normal	2011-02-24 20:05:22	us_er	Root
frankfurt	Normal	2011-02-24 20:05:22	de_er	Nonroot
singapore	Normal	2011-02-24 20:05:22	ap_er	Nonroot

Figure 3-10 Verification of the available replication server

10. Use the same approach to add all the other database servers into the replication network.

3.5.2 Creating the replication server using the cdr command-line utility

The cdr command-line utility provides a powerful interface to maintain and monitor all major ER components. The **`cdr define server`** command is the starting point to create a new replication server. Each new replication server must be specified locally by a separate call.

There are several options that can be specified for creating the replication server: the position of the server in a hierarchical replicate topology, the location for error files, timeouts for server connections, and the master server for catalog synchronization.

Root, non-root, and leaf servers

In the sample replication infrastructure defined in Figure 3-4 on page 39, the *newyork* server should be defined as the root replication server and the other server are treated as non-root servers. A possible change of the topology could be that frankfurt server is introduced as an intermediate level in the communication from newyork to the london server. In this case, the london server would be handled as a leaf server.

Using the hierarchy definition options provided by the **`cdr define server`** command, Example 3-2 shows possible combinations of the command options.

Example 3-2 Possible definitions of a replication server

```
cdr define server -I newyork
cdr define server -L -S newyork -I singapore
cdr define server -S newyork -N -I frankfurt
cdr define server -S frankfurt -L -I london
```

After defining the replication server, you can monitor the visibility of the server and the communication flow. Example 3-3 shows the output for our topology.

Example 3-3 Server visibilities and flow directions

```
$ cdr list server
SERVER                 ID STATE    STATUS     QUEUE  CONNECTION CHANGED
-----------------------------------------------------------------------
frankfurt            2934 Active   Connected      0 Feb 23 04:30:12
london               2936 Active   Connected      0 Feb 23 04:26:44
newyork              2930 Active   Local          0
singapore            2932 Active   Connected      0 Feb 23 04:20:51
```

ATS and RIS files

On the target side, the application of the replicated data might fail because of several reasons. There could be schema differences, timing issues, such as when an insertion row from one server is delayed but an later update from another server has already arrived, lock issues, and so on. The rejected data that cannot be applied because of conflict resolution policies defined later on in the replication has to be written to a ATS or RIS file.

These files are related to detected aborted transactions and contain detailed row information. The content of the files typically contains the time and the reason why the transaction application on the target replication server was aborted (specified by an SQL error). The RIS file also contains the details of the failed row for a later manual application or manual conflict resolution. Most of the time, the existence of the ATS or RIS files are indications about inconsistencies of the data for the defined replicate base tables. They could also be an indiction that there are schema differences between the source and destination tables in terms of defined constraints and column layout.

Here is a definition of a new server with the RIA and ATS files specified:

```
cdr define server -A /informix/ats -R /informix/ris -N -S newyork -I london
```

The specification of the ATS and RIS location is a global server setting. The DBA can decide, by using the definition of the replicates or the special settings of the configuration parameter CDR_SUPPRESS_ATSRISWARN, if, and which kind of, failures will be reported by the database server.

Timeout settings

The **cdr define server** command provides the option to define a timeout setting for the communication between the currently defined server and the connection server. This value, specified in minutes, is how long a server to server connection will be kept idle before the connection is closed.

If the connection is closed, attempting to send replication data between the server re-establishes the connection. A replication server with an idle timeout of one minute can be defined using the following command:

```
cdr define server -i 1 -N -S newyork -I frankfurt
```

3.5.3 Verification of success

You can verify the success of the **cdr define server** operation in various ways:

- Check if there are any error messages returned by the cdr utility.
- Check the existence of a new system database syscdr on the local database server.
- Check the log file for the sequence of messages shown in Example 3-4.

Example 3-4 Common log file output for a successful replication server definition

```
18:45:54  Building 'syscdr' database ...
18:45:54  dynamically allocated 2000 locks
18:45:55  Logical Log 7 Complete, timestamp: 0x210c2.
18:45:55  Loading Module <SPLNULL>
18:45:56  'syscdr' database built successfully.
18:45:57  CDR queuer initialization complete
18:45:57  CDR NIF listening on asf://newyork
18:45:57  ER checkpoint started
```

- Check the output of the **cdr list server** command for the newly added database server.
- Run **onstat -g ddr** and check if the relevant values for replication are initialized (Example 3-5).

Example 3-5 onstat -g ddr output

```
# Event  Snoopy   Snoopy   Replay   Replay   Current  Current
Buffers   ID      Position  ID      Position   ID     Position
```

```
1040     9        3696e0   9        34e018   9        36a000

Log Pages Snooped:
      From      From      Tossed
     Cache      Disk  (LBC full)
        26       848           0
```

3.5.4 Troubleshooting

If defining a local server as a replication server failed, check the settings. There are two major settings that affect the cdr defining server:

- Wrong or missing communication parameter definitions in the `sqlhosts` file
- Missing dbspaces required for the smart blobs in the send queue

Communication parameter

For the `sqlhosts` file, check the following settings:

- Did you made syntax errors?

  Check the connection string carefully. In this example, the definition failed because a "-g=" is used instead of the correct "g=":

  ```
  $cat sqlhosts
  singapore group          -                      - i=2934
  ap_er onsoctcp        100.3.1.1 23468 -g=singapore

  cdr define server -N -S newyork -I singapore
  command failed -- server not defined in sqlhosts  (35)
  ```

- Did you assigned your server to any group?

  If you defined a group of servers in the `sqlhosts` file, did you attach all your servers to the appropriate server group? In our example, we attach two servers to the same group but not to separate groups:

  ```
  singapore group          -                      - i=2932
  ap_er onsoctcp        100.3.1.1 23465 g=singapore

  frankfurt group          -                      - i=2934
  de_er onsoctcp        100.2.1.1 23468 g=singapore

  $> cdr define server -N -S newyork -I frankfurt
  command failed -- Enterprise Replication not active  (62)
  ```

- Did you use the group name for the definition? If you use the server name instead of the group name, the definition fails:

  ```
  frankfurt group          -                      - i=2934
  de_er onsoctcp        100.2.1.1 23468 g=frankfurt
  ```

```
   $> cdr define server -N -S newyork -I de_er
   command failed -- Enterprise Replication not active  (62)
```

Missing dbspaces

You are required to define a sbspace within the setting of the CDR_QDATA_SBSPACE configuration parameter. If the specified sbspace does not exist or is spelled wrong, the server definition fails and you see the following message:

```
$> cdr define server -N -S newyork -I singapore
command failed -- Sbspace specified for the send/receive queue does not exist
(107)
```

3.6 Special considerations for setting up replication

Here we provide guidelines for the values of the configuration parameters that you have to modify for the replication server. The main focus of our discussion is system performance. Here are the questions that you need to answer to set up the system properly:

- Are all the database servers in the replication required to be available all the time?
- Is there a need to keep data until the database server becomes available again?
- Which resource is more critical, memory or disk space?
- Do I have enough log space to keep the additional logs required for replication?
- Do I want to set up time based conflict resolution for replications? There are additional disk spaces required for the shadow tables and shadow columns.
- Do I have enough resources to generate a non-blocking data flow in terms of threads?

Answering the first two questions is essential. Are there any database servers associated with the replication that are frequently unavailable? Do you have to keep the replicated data for specific targets for a long period time? If the answer is “yes”, where do you want the data kept (on memory or on disk)?

3.6.1 Memory

In the normal replication scenario, the available memory should be large enough to keep all the data that cannot be replicated immediately. The size of the memory is ruled by the CDR_QUEUEMEM parameter. The optimal size of the memory pool depends on how much data is generated in which time. The queue size must be monitored. If spooling happens too fast, the memory must be adjusted. Pay attention to this memory value if the memory is set too low; data is spooled out to disk and the performance of the replication is affected by additional I/O operations for write and reads.

We discuss more details about monitoring enterprise replication in Chapter 6, "Monitoring and troubleshooting Enterprise Replication and Flexible Grid" on page 247.

3.6.2 Disk space requirements

ER requires additional disk space for standard dbspaces and for sblob spaces. This is particularly relevant in those cases where the system has to spool send or receive packages for specific targets, or when the memory definitions for internal ER related queues are too small. In general, the following list describes the major sources for disk space:

- Sycdr database creation
- Grouper page file
- Send queue spooling
- Receive queue spooling
- Shadow columns in update-anywhere and time stamp/SP conflict resolution
- Shadow tables in update-anywhere and time stamp/SP conflict resolution

CDR_DBSPACE, CDR_QHDR_DBSPACE, and CDR_QDATA_SBSPACE are configuration parameters that specify the disk space related information.

CDR_DBSPACE

The syscdr database is created at the time the replication server is defined, and contains all major maintenance information about the ER system. The dbspace where the database resides can be configured by an **onconfig** parameter. However, the specified dbspace has to exist at the time the replication server is defined. The default dbspace is the rootdbs. After being created, the syscdr database cannot be moved in a running ER environment.

CDR_QHDR_DBSPACE and CDR_QDATA_SBSPACE

The grouper threads are responsible for extracting data to be distributed from the processed transactions. This process is done based on the replicate definitions. If the defined memory for the processed transaction is exhausted, the transaction will be partially spooled into the system defined sblob space, which has to exist at replication server definition time.

Send queue spooling happens in two different cases:

- The memory for the send queue is exhausted because too many transactions are waiting to be sent out.
- The actual log position and the replay position of the last data from all server acknowledged transactions come too close for a log wrap-around.

The server stores the send queue header in a system defined regular dbspace. The actual transaction information is stored as a sblob in an sblobspace.

Receive queue spooling happens if the volume of incoming transactions on a target is much higher than the server can process. To keep them stored and prevent data loss, they are stored in a regular dbspace and in a sblobspace, similar to the send data.

Be aware that the administration goal should be to avoid filling completely the sbspace specified in the CDR_QDATE_SBSPACE parameter. Running out of space here causes the ER system to hang, which can typically only be resolved by restarting the server. To avoid that situation, the server raises an alarm. Following that alarm, a new chunk should be attached to the sbspace. The easiest solution to prevent that situation from happening is to find the bottleneck that caused the problem. It is most likely a particular target server.

dbspaces containing replication tables

Shadow columns have to be added to a table in case the table has to be used in an update-anywhere scenario with time stamp or SP conflict resolution. The additional disk space has to be calculated with 8 bytes per row in the base table. The disk space will be consumed in the same dbspace as where the table resides.

Shadow tables are automatically created for each table that is configured, similar to shadow columns, in a replication for update-anywhere and time stamp and SP conflict resolution. The table is created in the same dbspaces and with the same fragmentation schema as the master table. All deleted rows on base table are inserted into the shadow table for conflict resolution handling on late incoming changes, to avoid ghost rows. The contents of the delete table are frequently cleaned by the database server.

3.6.3 Logical log space estimates

Tables can be defined as a part of a replication set, which means that any changes on the table can be replicated to the target database server. Operations such as UPDATE statements executed on these tables require more log space than executed on the tables that are not replicated. There is a simple reason. When we run an update on a local table, Informix tracks only the changed values in the log entry because the database server can ensure that the row exists in the system where the update was executed. In a replication, the row might not be replicated to the target due to time constraints, slow performance, or a communication problem. Depending on the conflict resolution policy, we need to be able to apply the complete row within an insert instead running the update.

There is another requirement for additional log space. In case your memory pool for keeping the send data is configured too low, or one of the target database servers is unavailable and send data is spooled, these spools will also be kept in the log file.

Depending on how frequently update statements are used and how large the row sizes in the affected tables are, the logical log space must be adjusted.

4

IBM Informix Flexible Grid

Enterprise Replication (ER) can be used to create a multi-node heterogeneous cluster. However, basic ER is rather complex to use and historically only replicates changes to existing tables. It does not replicate such things as table creation, index creation, and stored procedure creation. Not only does this limitation increase the complexity of deploying a solution based on ER, but it also hampers our ability to run our applications on any node that is replicating data. It is not enough just to replicate the changes to table data. We have to, for example, replicate constraint rules, stored procedures, triggers, and so on, if we want to be able to run our applications on any node within the ER domain.

The Flexible Grid addresses these restrictions so that it becomes possible to run the application on any node within that grid. Not only are DML operations replicated, but so are DDL operations, which means that when table is created on one node, it is automatically created on all nodes within the Flexible Grid. In addition, it is possible to automatically start replication of the table as soon as it is created, which greatly simplifies the deployment of a replication solution. In the past, we had to create our tables on all of the node, then create all of the constraints, indexes, triggers, and stored procedures on each of the nodes. Then we had to define the replication rules for the table and finally start using the table. Now we simply create the schema on a single server, and immediately start using the table.

In this chapter, we explore Flexible Grid. We learn how to set up a grid, define the scope of the grid, use the grid, and troubleshoot any problem that might be encountered while using the grid.

4.1 Why use Flexible Grid

Use cases are often the best way to describe why functionality is useful. In this section, we present three use cases that describe the usefulness of Flexible Grid.

4.1.1 Case 1: Store inventory cleanup

Sue is the DBA for a large retail chain. Their corporate management has decided to retire a significant amount of old inventory in their stores, which includes such things as buggy-whips and horse carriages. They have never done an inventory cleanup in the past and still carry items that they do not have in inventory and see no market for those items.

Sue has estimated that there will be some four million items to be removed. She is concerned about this process because although the table is replicated, the network cost to replicate all of these deletes will be rather high. Also, she is not convinced that the inventory data in the stores perfectly matches the inventory data in the corporate system.

Corporate management has simplified the process a bit by agreeing to retire any product item that was introduced prior to the year 2000.

The solution

Fortunately, Sue has defined all of the stores to be part of a Flexible Grid. She can easily issue a cleanup of the inventory by executing a single statement (Example 4-1) at the central corporate system.

Example 4-1 Clean out your inventory with Flexible Grid

```
execute procedure ifx_grid_execute(‘stores_grid’,
   ‘delete from product_line where introduction_year < 2000’);
```

4.1.2 Case 2: New job classifications

Jim is a DBA at a large shipping company. He has been asked to create a new job classification so that each of the locations of his company can have a job classification for a person who manages the stock of hazardous materials. Although this is only a single row, there are several hundred servers that will need to have the new job classification so that they can hire personnel to manage the handling of those materials.

Jim has a personal issue with this task because he promised his son that he would act as coach for a softball team and practice begins in 30 minutes. Yet Jim’s boss is insisting that this get done within the next two hours.

Although Jim’s systems use Enterprise Replication to keep track of shipping workflow, the job_classification table is not replicated.

The solution

Jim attended the most recent Informix user group meeting where he learned about Flexible Grid. He decides to give it a shot. He is able to get this new row on each of his systems in just a few minutes by running the script shown in Example 4-2. He is able to get to baseball practice on time.

Example 4-2 Define Flexible Grid and run an INSERT statement

```
cdr define grid temp_grid --all
cdr enable grid -g temp_grid --user=jim --node=coporate_system

dbaccess hr@coporate_system <<!EOF
execute procedure ifx_grid_execute(‘temp_grid’,
    ‘insert into job_class values(‘hzard_stock’,....);
!EOF
```

4.1.3 Case 3: Performance analysis

Susan is the DBA for a hotel chain. She has been receiving complaints that it is taking a rather long time for customers to check out. Some customers have even missed airline flights because of the slowness of their checkout process. Her CEO is a bit miffed because of all of the complaints that they are receiving and especially about potential legal actions that is being taken by the people who missed their flights due to the slow checkout process.

Susan has done analysis of the problem and is fairly convinced that they can improve the performance by adding an index to the customer_charges table. There are several hundred hotels in the chain.

The solution

Susan has set all of the servers up as members of a Flexible Grid. Therefore, she is able to deploy the new index by simply creating the index on a local system. The Flexible Grid automatically deploys the new index on all servers within the grid in a matter of minutes. All Susan has to do run the statement shown in Example 4-3.

Example 4-3 Creating an index with Flexible Grid

```
dbaccess prop@coporate << !EOF
execute procedure ifx_grid_connect(‘hotels’, 1);
create index cust_chrgtype on customer_charges(customer_id, charge_id);
```

4.1.4 Case 4: The heterogeneous cluster

Larry has a cluster based on Enterprise Replication. Some of the servers are based on Linux and others are based on AIX. He wants to use load balancing across all of the nodes that are located in various cities across the nation.

One of the problems that he has is schema inconsistency. He has manually corrected these inconsistencies but wants the system to maintain them, which is an issue because the system is rather dynamic. New tables and stored procedures are created on a fairly regular basis. Larry is tired of having to maintain this cluster on a constant basis. Also, it is painful to constantly be having to define replication on the tables after the tables are created. He needs some relief of his efforts.

The solution

Larry implements Flexible Grid on his heterogeneous cluster. Then he implements a stored procedure that places the user automatically connected to the grid so that any new database objects such as tables or indexes, are automatically created on all the servers within the grid and replication is also automatically created. He accomplishes this task by executing the SQL script shown in Example 4-4.

Example 4-4 Using a stored procedure with Flexible Grid

```
dbaccess clustdb <<!EOF
execute procedure ifx_grid_connect(‘cluster_grid’, 1);
create procedure sysdbopen()
    execute procedure ifx_grid_connect(‘cluster_grid’, 3);
end proocedure;
```

This script causes all future DDL statements to be propagated to the other nodes in the Flexible Grid and also automatically establishes Enterprise Replication on any newly created table.

4.2 Establishing Flexible Grid

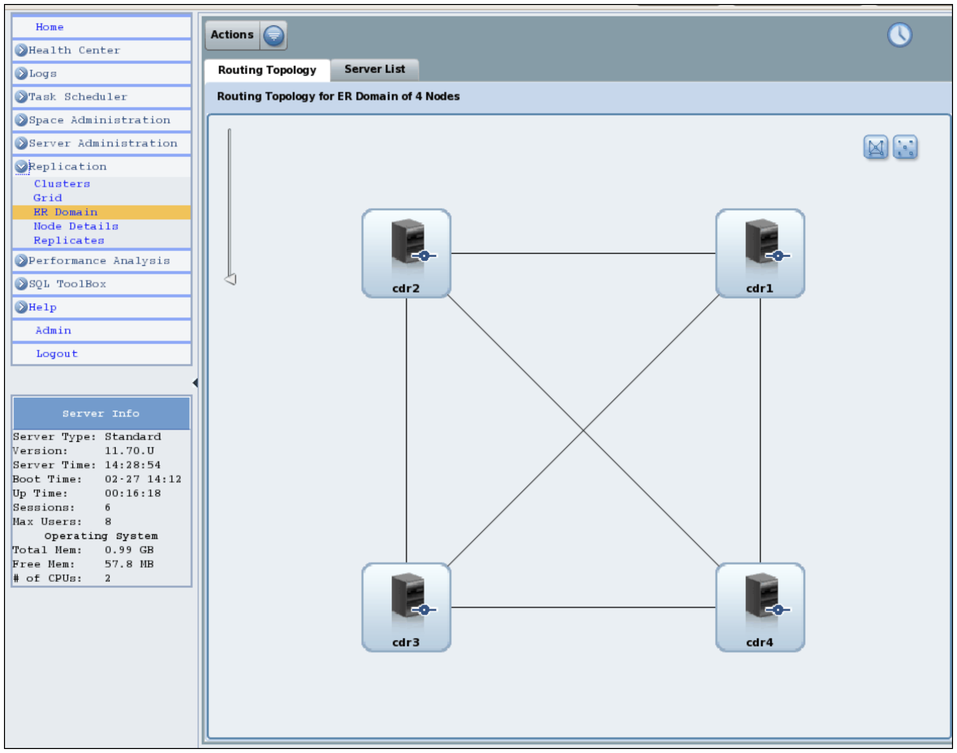
In the past, we defined the replication domain when defining the replicate. There was no pre-determined domain. When using Flexible Grid, we do the opposite. We first define the domain of Flexible Grid, then we create database objects within that grid.

In Chapter 3, “Configuring a server for Flexible Grid and Enterprise Replication environments” on page 31, we saw how to create the basic replication environment. We did not actually define any replication, but simply defined how the servers within the replication domain would communicate with each other. We now build on that foundation to define a Flexible Grid on those servers.

We use the Open Admin Tool (OAT) to define the Flexible Grid. As before, we also show the **`cdr`** command for the same operations.

4.2.1 Defining the servers within the Flexible Grid

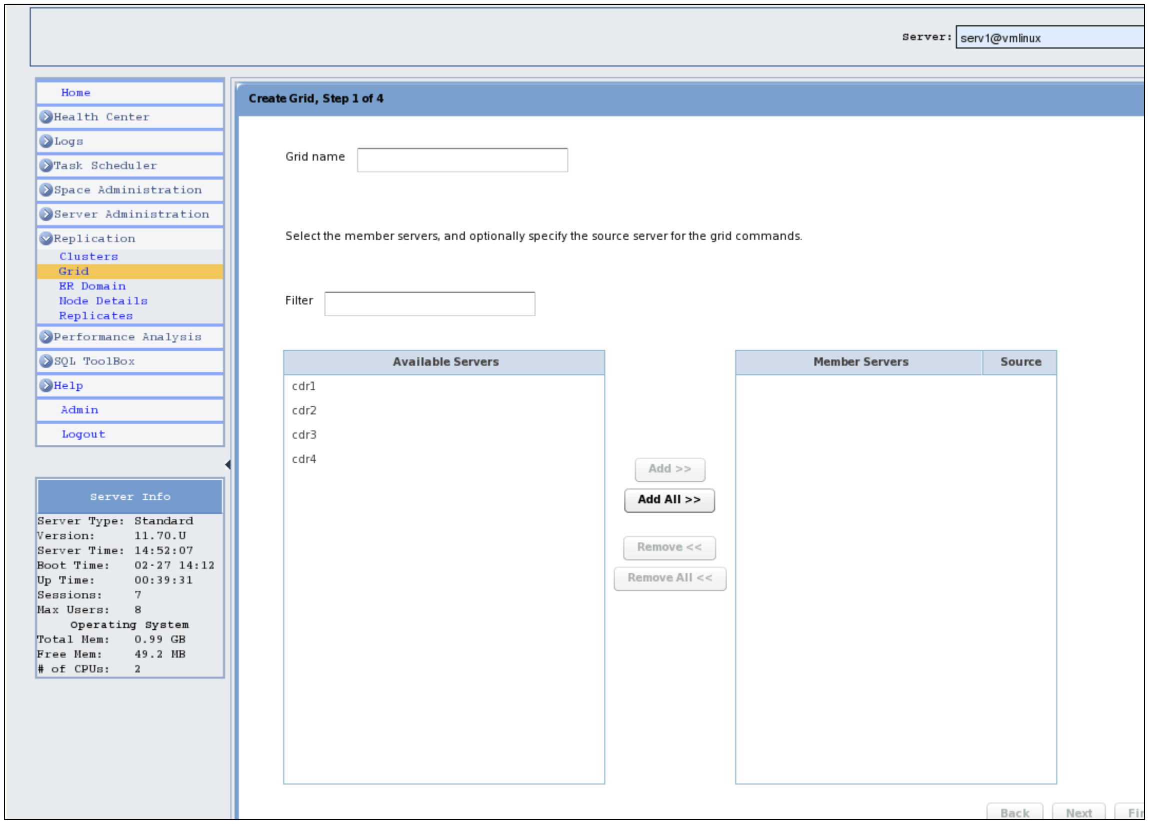
We first define the servers that will participate in the Flexible Grid. We can use OAT and navigate to the ER domain to view the servers that we have defined to participate within ER by selecting **Replication** → **ER Domain**. Figure 4-1 shows which servers are currently defined.



```
Server Info
Server Type:  Standard
Version:      11.70.U
Server Time:  14:28:54
Boot Time:    02-27 14:12
Up Time:      00:16:18
Sessions:     6
Max Users:    8
     Operating System
Total Mem:    0.99 GB
Free Mem:     57.8 MB
# of CPUs:    2
```

Figure 4-1 Exiting the ER domain

To define a Flexible Grid consisting of these four servers, navigate to the Grid page within OAT by selecting **Replication** → **Grid**. Because we have not yet defined a Flexible Grid, the initial display is empty. To define our grid, navigate to the Grid Create Screen by selecting **Actions** → **Create Grid**. The window shown in Figure 4-2 opens.



Select the member servers, and optionally specify the source server for the grid commands.

cdr1
cdr2
cdr3
cdr4

```
Server Info
Server Type:  Standard
Version:      11.70.U
Server Time:  14:52:07
Boot Time:    02-27 14:12
Up Time:      00:39:31
Sessions:     7
Max Users:    8
     Operating System
Total Mem:    0.99 GB
Free Mem:     49.2 MB
# of CPUs:    2
```

Figure 4-2 Create Grid initial window

This window allows us to name the grid and include the servers that participate within that grid. The name of the grid must be unique and become a replicate set for any table that is replicated as part of the grid. We include all servers within the Flexible Grid and name our grid “grid1”.

The grid’s name is used to also generate a replicate set of the same name. It is worth noting that any table that is created as a grid operation and is replicated becomes part of that replicate set.

The Add button is currently disabled. It becomes enabled when we highlight a server by clicking its name, as shown in Figure 4-3.

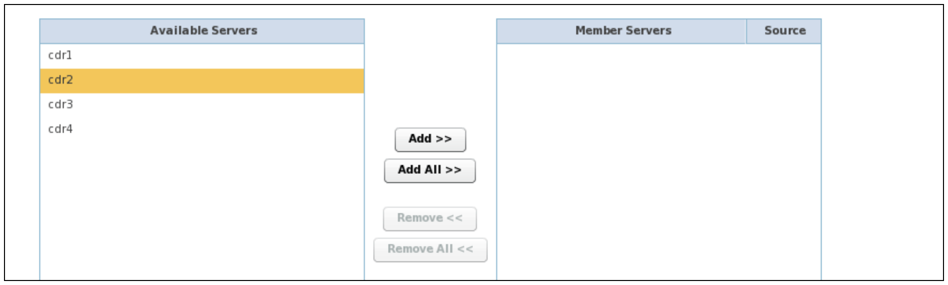
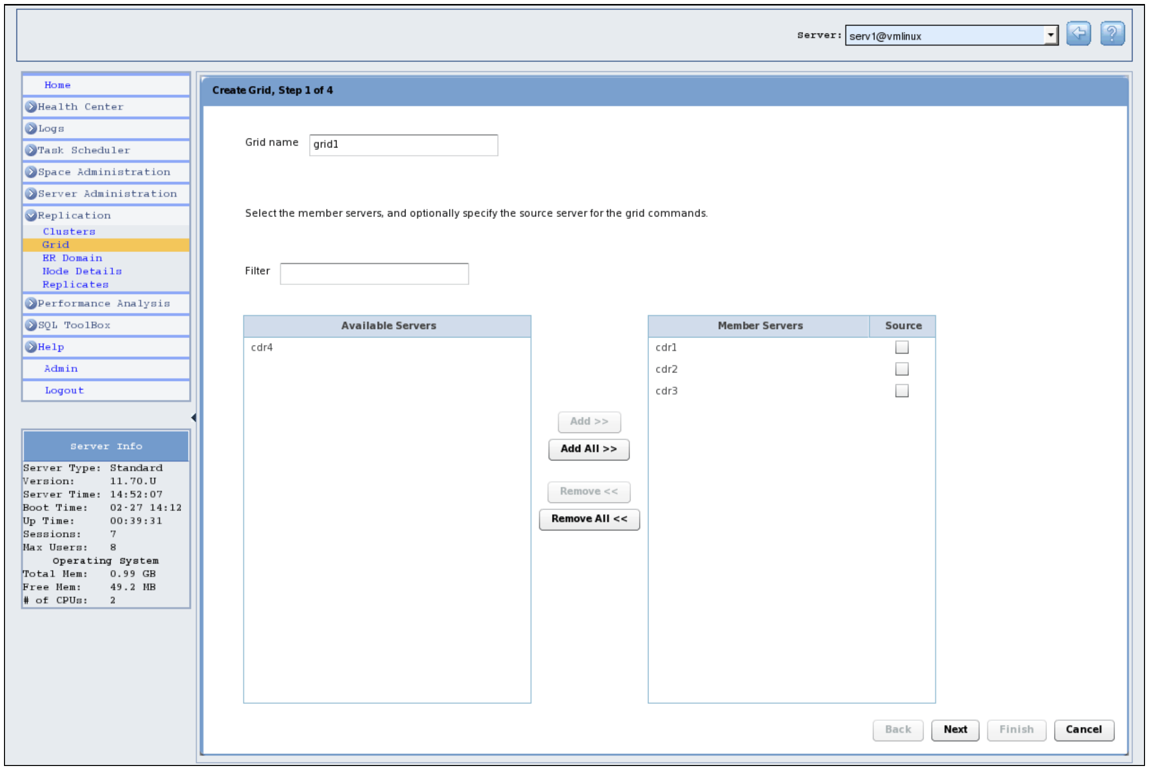


Figure 4-3 Adding nodes to the grid

For our purposes, we want to initially include servers cdr1, cdr2, and cdr3 within our grid. After adding these servers to the grid, we should see something like Figure 4-4.



```
Server Type:  Standard
Version:      11.70.U
Server Time:  14:52:07
Boot Time:    02-27 14:12
Up Time:      00:39:31
Sessions:     7
Max Users:    8
    Operating System
Total Mem:    0.99 GB
Free Mem:     49.2 MB
# of CPUs:    2
```

Create Grid, Step 1 of 4

Grid name: grid1

Select the member servers, and optionally specify the source server for the grid commands.

Filter:

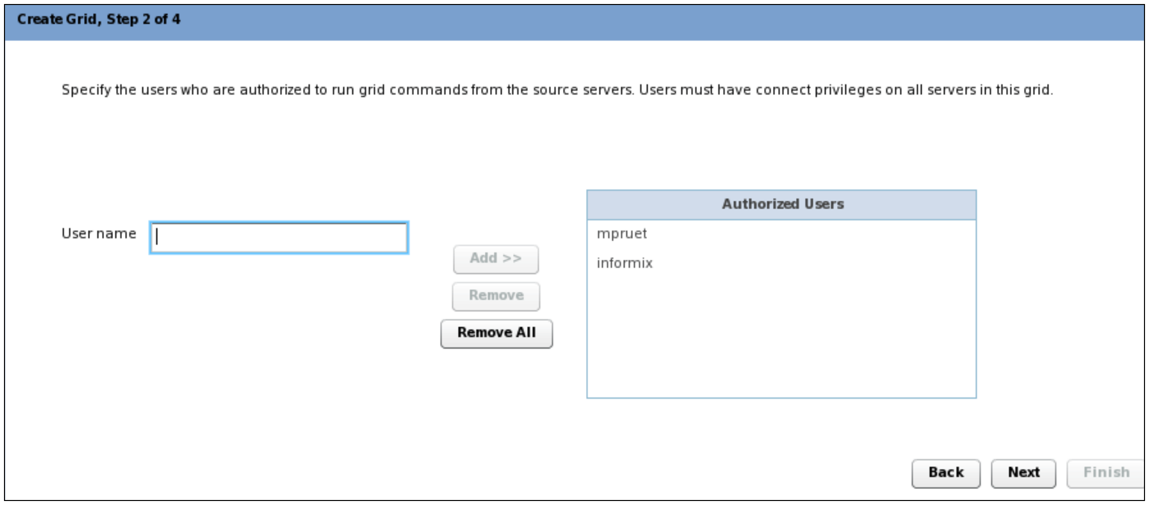
Available Servers
cdr4

Member Servers	Source
cdr1	☐
cdr2	☐
cdr3	☐

Figure 4-4 Setting servers for the grid

You might notice in this example that there is a Source button on each of the servers in the select list. This button is used to restrict on which servers grid oriented operations can be performed. We do not want to allow any of the servers within the grid to perform grid operations because that would allow a rogue DBA to drop tables on all nodes within the grid. For our purposes, we perform all grid operations on the cdr1 server, so we click **Server** and **Next** at the bottom of the page.

The next window provides a further restriction on grid operations. Not only do we want to restrict the servers from which a grid operation to be performed, but we also want to restrict the users that are allowed to perform a grid operation. By default, no user, including the informix user, is allowed to perform a grid operation.We will now allow the informix and mpruet users to perform grid operations. We do this by typing in the user names and then clicking **Add** for each user. At this point, the window should look like Figure 4-5.



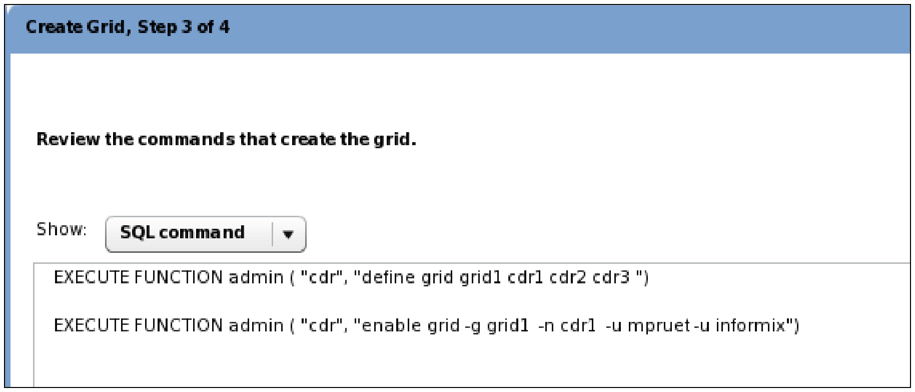
Specify the users who are authorized to run grid commands from the source servers. Users must have connect privileges on all servers in this grid.

Authorized Users
mpruet
informix

Figure 4-5 Allowing users to perform grid operations

We need to understand what is meant by enabling a source server and authorizing a user to perform grid operations. This task is done to limit the users and nodes that can create a grid object, such as a table or index that is propagated to the nodes within the grid. That grid operation could include the automatic enabling of the replication of the data that is inserted or changed into a table. The user of the table does not need to be authorized as a grid user. If that user should insert a new row into the table, it is propagated. However, the user will not be able to create a new table that is automatically propagated across the grid.

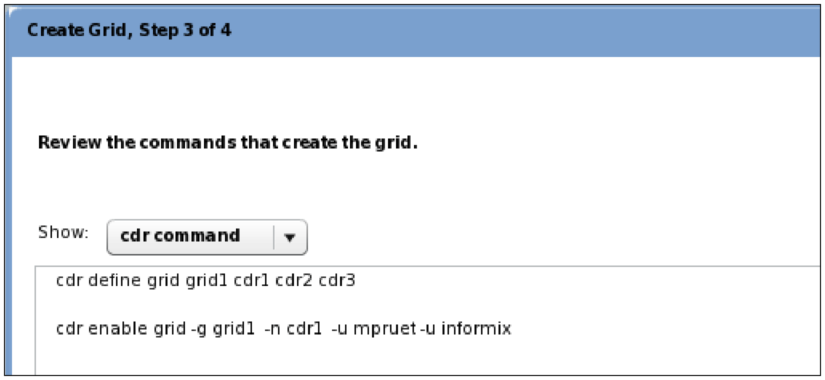
The next windows shows the commands that OAT executes to define the grid1 grid. We can display the SQL command that is performed by using the sysadmin database (Figure 4-6).



```
EXECUTE FUNCTION admin ( "cdr", "define grid grid1 cdr1 cdr2 cdr3 ")

EXECUTE FUNCTION admin ( "cdr", "enable grid -g grid1  -n cdr1  -u mpruet -u informix")
```

Figure 4-6   SQL commands used to define the grid

We can also view the commands that would be used by the cdr utility by selecting the down arrow next to the SQL command tab and then selecting **cdr command** (Figure 4-7).

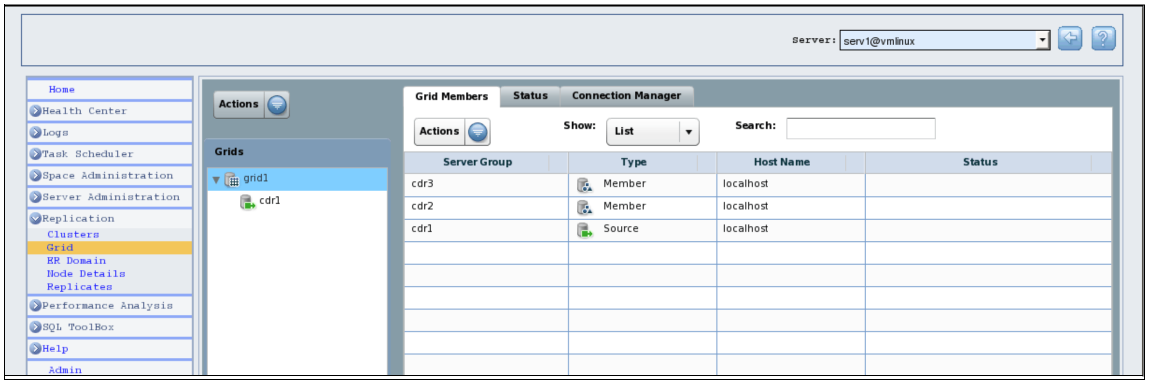


```
cdr define grid grid1 cdr1 cdr2 cdr3

cdr enable grid -g grid1  -n cdr1  -u mpruet -u informix
```

Figure 4-7   cdr utility commands used to define the grid1 grid

Click **Finish** to complete the creation of the grid.

After the grid creation completes, we can return to the main grid window. If we highlight the grid1 grid, we can see the members of the grid (Figure 4-8).



Server Group	Type	Host Name	Status
cdr3	Member	localhost	
cdr2	Member	localhost	
cdr1	Source	localhost	

Figure 4-8   View of grid members

4.2.2 Adding a new server to an existing Flexible Grid

In 4.2.1, “Defining the servers within the Flexible Grid” on page 62, we demonstrated how we could create a Flexible Grid using the cdr1, cdr2, and cdr3 servers. But we did not include cdr4 in the list of servers for the grid. We now want to add that server as a member of the grid.

To add a new server to an existing Flexible Grid, complete these steps:

1. Navigate to the Grid page by selecting **Replication** → **Grid**.
2.)Highlight the grid1 gird and click the **Actions** drop-down list on the Grid Members tab, as shown in Figure 4-9.

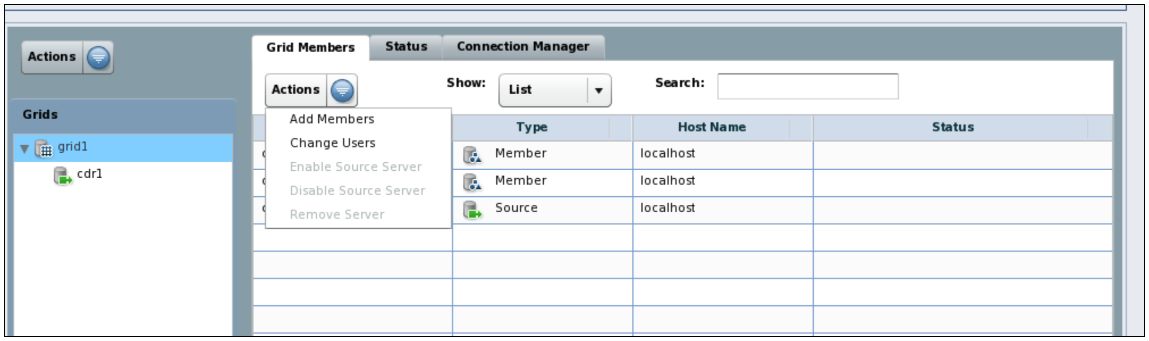


Figure 4-9   Add new member

3. Select **Add Members** to add the cdr4 server to the grid1 grid. Highlight **cdr4**, click **Add**, and then **Next** (Figure 4-10).

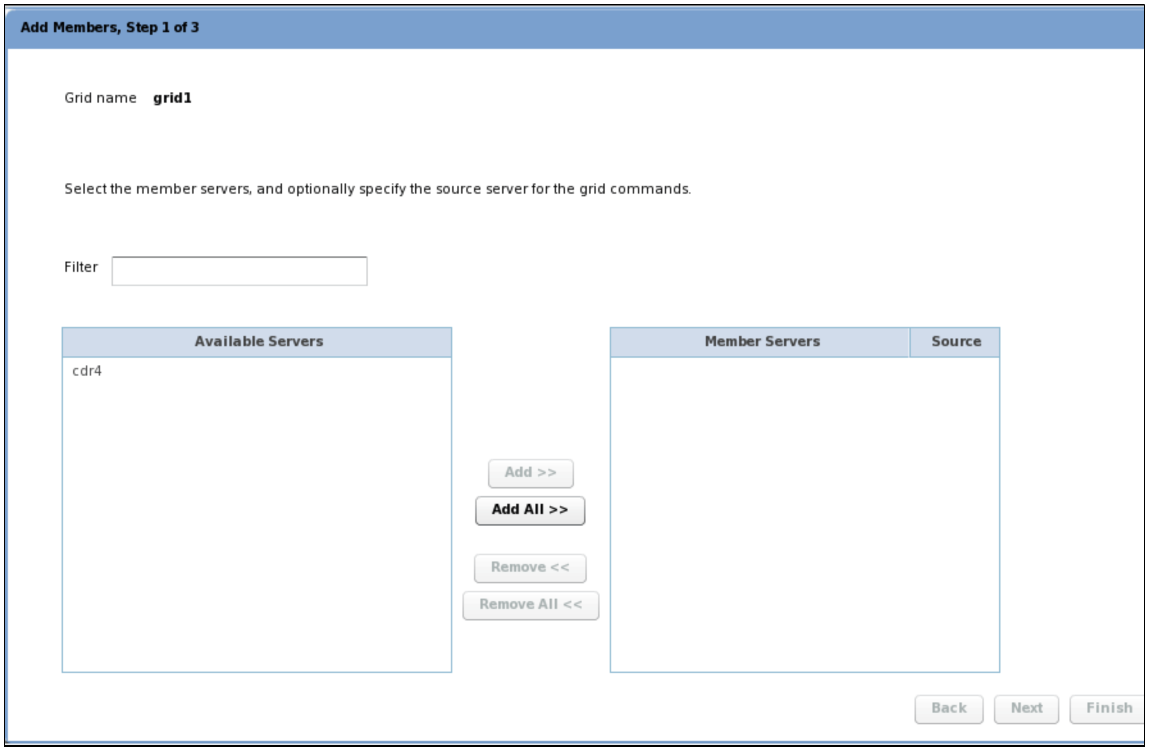
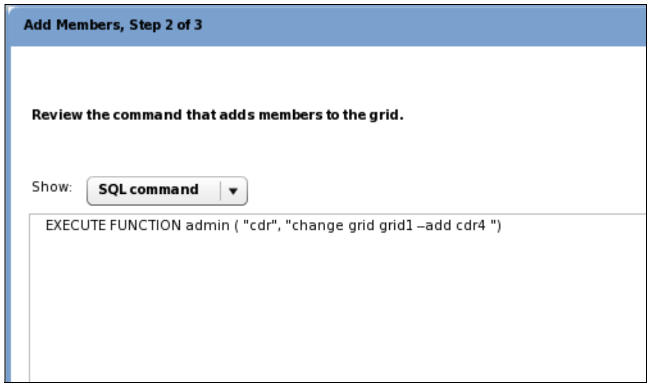


Figure 4-10 Adding a new node to the grid

4. The next window displays either the SQL or cdr utility command to change the grid (Figure 4-11). The similar **`cdr`** command would be **`cdr change grid grid1 --add cdr4`**.



```
EXECUTE FUNCTION admin ( "cdr", "change grid grid1 –add cdr4 ")
```

Figure 4-11 Command to add members

5. Click **Finish** to perform the operation. After the operation is completed, click **Done** to return to the main Flexible Grid window.

4.2.3 Enabling a node on an existing grid

We previously saw that we could restrict grid operations to specific members of the grid as a security precaution. There are occasions where we might need to extend that functionality to other nodes within the grid. For example, suppose that the normal server that we used as the source of grid operations was down. We might want to temporarily use some other node as a source of grid operations, which is done by enabling a node.

To enable a node, complete the following steps:

1. From OAT home page, navigate to the grid member page by selecting **Replication** → **Grid** and then click the grid name.

2. Highlight the server to be modified. We plan to allow the cdr2 server to become a source of grid operations, so we highlight the cdr2 server and click the **Actions** drop-down menu (Figure 4-12).

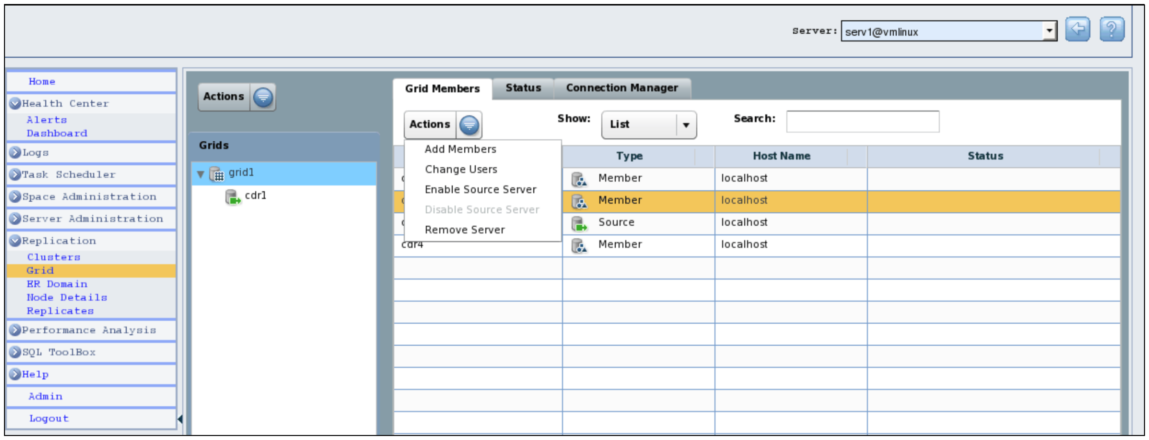
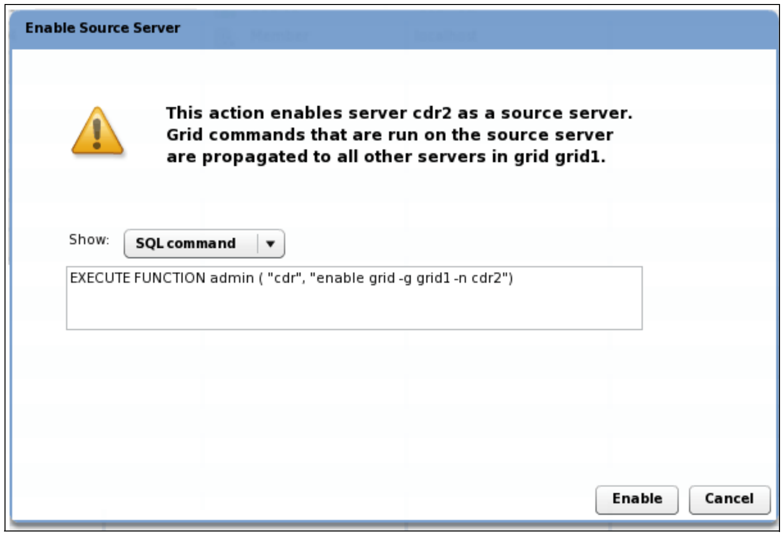


Figure 4-12 Enabling an existing member of the grid to be the source of grid operations

3. Click **Enable Source Server** to enable the node that has been highlighted.
4. The Enable Source Server window displays the command that is about to be executed (Figure 4-13).


Enable Source Server

This action enables server cdr2 as a source server. Grid commands that are run on the source server are propagated to all other servers in grid grid1.

Show: SQL command

```
EXECUTE FUNCTION admin ( "cdr", "enable grid -g grid1 -n cdr2")
```

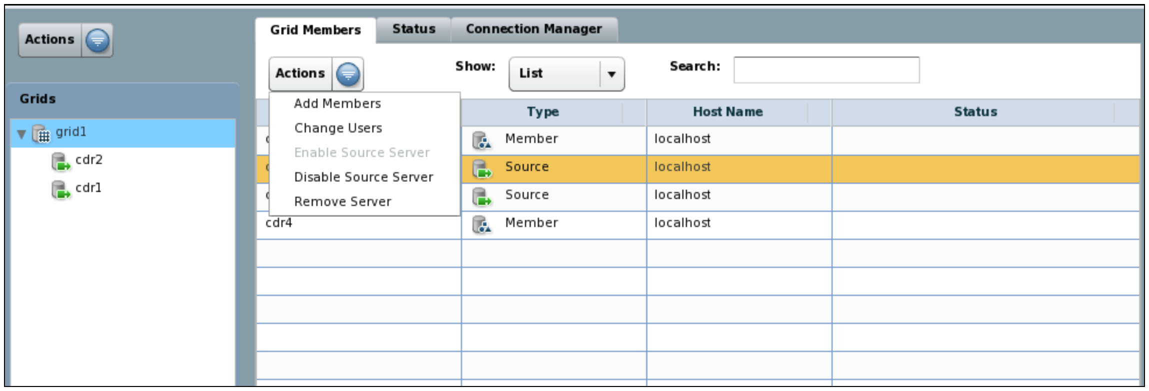
Figure 4-13 Confirmation of the enable grid command

5. Click **Enable**, and then **Close** on the subsequent window. We are taken back to the main Flexible Grid window, where we notice that the cdr2 server is now a source.

4.2.4 Disabling a node on the Flexible Grid

When we disable a node on the Flexible Grid, we block it from being used as a source for grid operations. As an example, we remove the cdr2 server as a source on the grid. To accomplish this task, complete the following steps:

1. From the main Flexible Grid window (select **Replication** → **Grid** to navigate there if it is not already open), highlight the grid, in our example, grid1.
2. Highlight the server to be removed as the source of grid operations and select **Actions** in the Grid Members tab (Figure 4-14).



	Type	Host Name	Status
c	Member	localhost	
	Source	localhost	
c	Source	localhost	
cdr4	Member	localhost	

Figure 4-14 Preparing to disable a node in the Flexible Grid

3. Click **Disable Source Server**.
4. The window that displays the command to be run opens. Click **Disable**, then **Close** in the subsequent window to return to the main Flexible Grid window. We notice that the cdr2 server is no longer a source.

4.2.5 Adding or dropping an authorized user

Adding a user as an authorized user means that the user can create grid objects on enabled servers. When the user is authorized for grid operations, that user can perform grid operations at any node that is enabled within the grid. The user's privilege is not restricted to any specific server.

We can either highlight the grid name or a server within the grid to authorize a user. To authorize a user for grid operation, complete the following steps:

1. From the main Flexible Grid window, highlight the grid1 grid and select the **Actions** drop-down menu.
2. Click **Change Users**. This action opens the window that we saw when we created the grid and entered the users who were authorized to perform grid operations (see 4.2.1, "Defining the servers within the Flexible Grid" on page 62).

4.2.6 The cdr commands for the management of a Flexible Grid

We use the Open Admin Tool (OAT) to create and manage the Flexible Grid. You also can use the cdr utility directly to create the grid and might find it useful to do that if you need to script the creation as part of an embedded deployment. The commands are documented thoroughly in *IBM Informix Enterprise Replication Guide*, SC27-3539. We briefly describe each of the commands here:

- To define a grid, run:

  ```
  cdr define grid <grid_name> <node1> <node2> <node3> ...
  ```

- To delete a grid, run:

  ```
  cdr delete grid <grid_name>
  ```

- To add a node to an existing grid, run:

  ```
  cdr change grid <grid_name> --add=<node_name>
  ```

- To remove a node from a grid, run:

  ```
  cdr change grid <grid_name> --del=<node_name>
  ```

- To enable a node as the source of grid operations, run:

  ```
  cdr enable grid --grid=<grid_name> --node=<node_name>
  ```

- To disable a node as a source of grid operations, run:

  ```
  cdr disable grid --gird=<grid_name> --node=<node_name>
  ```

- To authorize a user to perform grid operations, run:

  ```
  cdr enable grid --grid=<grid_name> --user=<user_name>
  ```

- To remove authorizations for a user to perform grid operations, run:

  ```
  cdr disable grid --grid=<grid_name> --user=<user_name>
  Using a Flexible Grid
  ```

In 4.2, "Establishing Flexible Grid" on page 61, we learned how to create a Flexible Grid. In this section, we discover how to use the Flexible Grid after we have defined the servers that make it up.

In 4.2, “Establishing Flexible Grid” on page 61, we tried to describe what we meant by a “grid operation”. In this section, we explore the concept of a grid operation a bit further. Fundamentally, a grid operation is an operation in which the operation is executed on all of the nodes within the grid. This is a bit different from the propagating of data changes that are part of Enterprise Replication. With ER, we do not replicate the DML statement. The original DML statement is not executed on the various nodes within ER (or within the grid). With a grid operation, the execution of the operation itself is propogated.

As an example, consider the server-a and server-b. We have a table called tab1 on both servers with the col1, col2, col3, col4, and col5 columns. Table 4-1 and Table 4-2 shows the rows of tab1 in server-a and server-b.

Table 4-1 Rows in tab1 on server-a

col1 (key)	**col2**	**col3**	**col4**	**col5**
1	cat	purr	24	20
2	cat	nip	25	10
3	bird	tweet	8	16
4	cat	ate bird	2	91

Table 4-2 Rows in tab1 on server-b

col1 (key)	**col2**	**col3**	**col4**	**col5**
1	cat	purr	24	20
2	cat	nip	25	10
3	bird	tweet	8	16
4	dog	bark	2	91

If we execute the `update tab1 set col3 = “fur” where col2 = “cat”` statement and are using basic Enterprise Replication, then we would end up with the results shown in Table 4-3 and Table 4-4.

Table 4-3 Rows on tserver-a after execution

col1 (key)	col2	col3	col4	col5
1	cat	fur	24	20
2	cat	fur	25	10
3	bird	tweet	8	16
4	cat	fur	2	91

Table 4-4 Rows on server-b after replication

col1 (key)	col2	col3	col4	col5
1	cat	fur	24	20
2	cat	fur	25	10
3	bird	tweet	8	16
4	cat	fur	2	91

Notice that the row with a key of 4 changed from a dog to a cat. This is because we replicated the result of the replication, not the statement that was executed.

If we had performed the update statement as a grid operation, then the result in server-b would be as shown in Table 4-5.

Table 4-5 Rows on server-b after grid operation propagation

col1 (key)	col2	col3	col4	col5
1	cat	fur	24	20
2	cat	fur	25	10
3	bird	tweet	8	16
4	dog	bark	2	91

The reason for the difference is that the latter replicated the execution of the statement rather that the replication of the result of the execution.

Normally, we use basic ER to replicate the data changes on tables even if we taking advantage of the Flexible Grid. We might create a table as a grid operation and use auto-registration to define that table as a replicated table. However, there might be reasons to occasionally perform maintenance operations as a grid operation. For example, we might need to delete all rows over a certain age.

We could chose to simply delete the rows, but that would mean replicating the effect of a large amount of deletes, perhaps several million deletes. It might be better to simply execute a delete statement as a grid operation, which would require much less network traffic. By default, the result of a grid operation is not in itself replicated, so there would be no additional network traffic as the result of the execution on the remote servers.

4.2.7 Connecting to the grid

By connecting to the grid, we are stating that any DDL statement that we issue will be propagated across the grid. We can also specify that replication will be automatically created on those DDL objects.

To connect to the grid, use the built-in **`ifx_grid_connect()`** command. The format of the command is:

```
ifx_grid_connect(<GRID_NAME>, <MODE>, <TAG>);
```

Where:

- GRID_NAME is the name of the Flexible Grid that we want to propagate the DDL operations across. It is a character string and must be the first parameter.
- MODE is a integer that describes if automatic registration is to be performed. The possible values are:
  - 0: Replicate only the DDL without performing auto-registration with ER. Return an error if executed by a non-authorized user. This is the default mode.
  - 1: Replicate the DDL and auto-register any table as an ER replicated table. Return an error if executed by a non-authorized user.
  - 2: Replicate only the DDL without performing auto-registration with ER. Ignore the request with no error returned if executed by a non-authorized user.
  - 3: Replicate the DDL and perform auto-registration with ER. Ignore the request with no error returned if executed by a non-authorized user.
- TAG is an optional character string that can be used to make it easier to monitor the status of the command.

The only required parameter is the grid name. The mode and tag are optional parameters and can be left out.

The order of the last two parameters can vary. We use the fact that *mode* is an integer and *tag* is a character string to correctly process the parameters. The following lines are equivalent:

```
execute procedure ifx_grid_connect('grid1', 2, 'myTag');
execute procedure ifx_grid_connect('grid1', 'myTag', 2);
```

Because the mode and tag parameters are optional parameters, the following lines are equivalent:

```
execute procedure ifx_grid_connect('grid1');
execute procedure ifx_grid_connect('grid1', 0);
```

Using the Flexible Grid for schema deployment

There are cases where we only want to deploy schema changes across multiple nodes, for example, when we have multiple servers in retail stores and need to deploy a new application. We have limited resources for the deployment and want to be able to make the schema changes across hundreds or even thousands of systems in each of our stores, as shown in Figure 4-15.

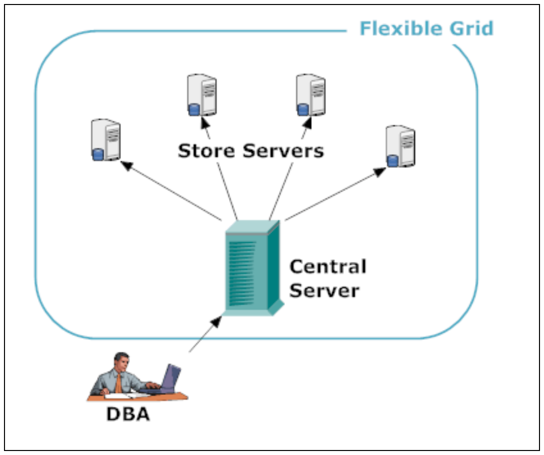


Figure 4-15 Using the Flexible Grid for schema deployment

The DBA only has to perform the DDL deployment once. From there, the DDL statements are propagated to all of the nodes within the Flexible Grid as a grid operation. This action minimizes the effort of the DBA so that he only has to perform the deployment once. Also, the DBA does not have to adjust his deployment as new stores are opened.

As an example of how this process would work, consider the SQL script in Example 4-5.

Example 4-5 Using DDL propagation without ER auto-registration

```
-- Connect to sysmaster. We do this first because we need to
-- execute ifx_grid_connect. It is a built-in procedure, and exists
-- even in the sysmaster database.

database sysmaster;
execute procedure ifx_grid_connect('grid1', 0);

-- Create an unlogged database
create database my_db;
create table pets(
   owner_id     integer,
   pet_age        integer,
   pet_type       char(10),
   pet_name     char(40),
   pet_weight    integer);


create index idx1 on pets(owner_id);

create function num_cats(age integer)
create function num_cats(age integer)
     returning integer
     define tnum integer;
     select count(*) into tnum
     from pets where pet_type = 'cat' and pet_age >= age;
    return tnum;
end function;
```

In this script, we complete the following steps:

1. We start by connecting to the sysmaster database. The ifx_grid_connect procedure is a built-in stored procedure that exists in all databases, but only within a database. It does not exist if you are not connected to a database. It is not possible to execute the stored procedure until we are actually connected to a database. We could have connected to any database, but used the sysmaster database because it is found on all Informix instances.
2. We connect to the grid1 grid with a mode of zero, which indicates that we want to propagate the DDL but do not want to automatically create replication on data changes for any tables that are created. We could have also simply executed **`execute procedure ifx_grid_connect('grid1')`**, because zero is the default mode.

3. We create an unlogged database. This database creation is propagated to all of the nodes within the grid1 grid because we issued **`ifx_grid_connect`** to that grid.
4. We then create a table, index, and stored procedure, which will also be created on the other nodes within the grid.

Considerations for schema deployment

When using the Flexible Grid for schema deployment, we must be aware that the application of the DDL on the various nodes within the grid is done under the same user ID as on the source node, and with the same LOCALE. Therefore, it is important that the user ID used to execute the grid operations be a valid user ID on all nodes within the Flexible Grid.

Using the Flexible Grid with auto-registration

Often, when we have a multi-node grid, we want to include data change replication as well as schema changes. We are using multiple nodes, such as a cluster, and thus want the data to be propagated as well as the schema changes. See Figure 4-16 for an example.

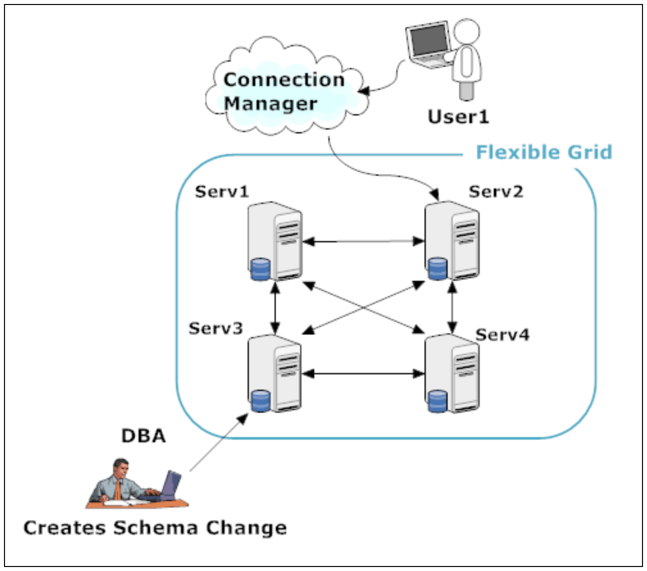


Figure 4-16 Using Flexible Grid with auto-registration

When the DBA creates a schema change to create a new table, then user1 is able to see those changes in just a matter of seconds. When user1 makes any modification to the new table, those changes are replicated to the other servers in the Flexible Grid.

Now let us examine a different SQL script to see how this process would work (Example 4-6).

Example 4-6 Using Grid DDL propagation with ER auto-registration

```
database sysmaster;
execute procedure ifx_grid_connect('grid1', 1);

create database my_db with log;

create table owners(
   owner_id                  serial   primary_key,
   owner_fname            char(20),
   owner_mi                 char(1),
   owner_lname            char(20),
   owner_phone            varchar(32),
   owner_phone_ext      varchar(12),
   owner_addr1             varchar(80),
   owner_addr2      varchar(80),
   owner_city                varchar(80),
   owner_st                  char(2),
   owner_zip1               integer,
   owner_zip2               integer
) lock mode row;

create table pets(
   owner_id                  integer,
   pet_age                   integer,
   pet_type                  char(10),
   pet_name                  char(40),
   pet_weight                integter);

alter table pets add constraint
     (foreign key (owner_id) references owners
     on delete cascade;

create function num_cats(age integer)
     returning intake
     define tnum integer;
     select count(*) into tnum
        from pets where pet_type = 'cat' and pet_age >= age;
    return tnum;
end function;

insert into owners (owner_fname, owner_mi, owner_lname, owner_phone,
                    owner_phone_ext, owner_addr1, owner_addr2, owner_city,
                    owner_st, owner_zip1, owner_zip2)
```

```
values (‘John’, ‘C’, ‘Green’, ‘9723443432’,null,’3504 Piney Point’, ’Delmar’,
        ’TX’,0,0);
```

In this script, we complete the following steps:

1. We start by attaching to the sysmaster database so that we can issue the **ifx_grid_connect** command. However, in this example, we pass a mode of 1, which signifies that we want to perform auto-registration of any table that we might create.
2. We create the my_db, with logging database. We need to make this a logged database, because otherwise we would not be able to replicate the data changes made to any tables within the database.
3. We create two tables, owners and pets. Because the ifx_grid_connect call was made with the mode set to 1, data changes to those two tables will be performed as well as the propagation of the creation of the tables.
4. Because the ER replication of the tables was a result of a grid operation, then those two tables are automatically included in the grid1 replicate set as well.
5. It is worth noting that although the owners table has a primary key, the pets table does not. In the event that there is no primary key that can be used by replication, then the server automatically generates a hidden column that ER can use as a key.
6. At the end of the script, we insert a row into the owners table. Because we are using auto-registration, this data changes from the insert statement are replicated as well.

Auto-registration summary

When a table is auto-registered, it is a strictly mastered replicate. When the table is altered, the table is automatically remastered so that the replication of the data changes continue while the DDL propagation is in-flight to the target nodes. If the table that was created by auto-registration is deleted, then the replicate definition is automatically deleted as well.

The name of the auto-registered replicate has the following format:

```
G<grid_id>_<source_id>_<stmt_number>_<table_name>
```

Where the fields have the following meanings:

- grid_id is a number that identifies the grid.
- source_id is the group_id from the `sqlhost` file.
- stmt_number is a number that identifies the statement.

Thus, if we perform auto registration of the “cats” table, then the replicate name might be something like *G11_10_1023_cats*. The replicate is included in the replicate set with the same name as the grid.

The auto-registered table might or might not have a primary key. If it does not, then Informix uses an internally generated key to identify the row. The cdr utility is aware of the internal key, which means that tools such as cdr check can be used with auto-registered tables.

4.2.8 Automating the grid connection

In many cases, we want to always be connected to a grid by default. We want to make it so that when we connect to the database that we can be assured that any database object that we create is automatically propagated to the other nodes within the grid. We can take advantage of the *sysdbopen* stored procedure to make this action occur.

The sysdbopen procedure is a special stored procedure that is executed when any user opens a database. We can perform the execution of the ifx_grid_connect procedure from within the sysdbopen procedure. The result would be to make the default so that we would propagate the creation of database objects and also to automate the creation of an ER replicate.

Example 4-7 shows how the sysdbopen stored procedure might be defined. We must use 3 as the mode when executing the ifx_grid_connect stored procedure within the sysdbopen procedure because the sysdbopen procedure could be executed by anyone. We do not want to return an error in those cases.

Example 4-7 Creating a sysdbopen procedure to automatically connect to a grid

```
database sysmaster;
execute procedure ifx_grid_connect(‘grid1’, 1);
create database mydb with log;
create procedure sysdbopen()
   execute procedure ifx_grid_connect(‘grid1’, 3);
end procedure;
```

If we later execute Example 4-8, we see that the tables created are also created on the other servers within the grid and that replication is automatically established. After performing the insert, we find that the data exists on all nodes within the grid1 Flexible Grid.

Example 4-8 Creating a table within the grid

```
database mydb;
create table mytab (
    myid                integer,
    mydata           varchar(100)) with crcols lock mode row;
insert into mytab (myid, mydata)
    values (1,'way cool');
```

4.2.9 Disconnecting from the Flexible Grid

Although we may normally use the Flexible Grid to automatically replicate the schema and automatically register new tables with ER, there are times when we might want to create a table that is local to the server. We can execute the ifx_grid_disconnect procedure to make this process happen. The ifx_grid_disconnect procedure does not have any parameters. Example 4-9 gives an example of this procedure.

Example 4-9 Using the ifx_grid_disconnect procedure

```
database salesdb;
execute procedure ifx_grid_connect('mygrid', 1);

-- These commands will be executed as a grid operation. They will be
-- propagated to all servers within the grid 'mygrid' and replication
-- of data changes will be automatically registered.
create table replicated_tab (
    cust_id                 integer,
    cust_lname          varchar(80),
    cust_mi                char(10,
    cust_fname          varchar(80));

-- Disconnect from the grid
execute procedure ifx_grid_disconnect();

-- This table is local to this database because
-- we are disconnected from the grid.
create table local_tab (
    local_id                 integer,
    local_cust_id        integer,
    local_select_cd    char(1),
    local_err_cd          integer);
```

4.2.10 Performing a grid statement execution (ifx_grid_execute)

There are times that we need to perform administrative work on the nodes within a Flexible Grid. These types of operations are generally outside of the main operations performed, but are still critical to normal operations. We can use the Flexible Grid to simplify this process.

As an example, suppose that we have a replicated table on all of the nodes within a 50 nodes grid. The table has several million rows. We have to remove all of the rows that are over year old. We could use basic ER to perform the removal of these rows, but that would require that we replicate over a million rows over 50 nodes. This would create a significant amount of network traffic. A better solution would be to propagate the execution of the delete statement to all of the nodes within the grid rather to propagate the changes made by the delete statement.

The format of the grid execution is as follows:

```
EXECUTE PROCEDURE ifx_grid_execute(<grid_name>,<statement>,<tag>);
```

Where the parameters have the following meanings:

- grid _name is the name of the grid.
- statement is the statement to be executed. It is a character string and must not contain “bound” data.
- tag is an optional string that we can to monitor the results of the execution.

Example 4-10 describes how we can use **ifx_grid_execute** to remove all rows older than a certain date. We do not have to perform the execution on all nodes within the grid individually, and we do not care what those nodes are.

Example 4-10 Using a grid execution to remove old rows from a table

```
EXECUTE PROCEDURE ifx_grid_execute (‘grid1’, |
    ‘delete from sales where sales_year < 2010’,
    ‘delete_sales_tag’);
```

The results of a statement that is executed as part of **ifx_grid_execute** are not replicated by ER.

4.2.11 Performing a grid function execution (ifx_grid_function)

It is possible to execute a function as a grid operation, which is particularly useful if you have to perform one of the sysadmin functions in the sysadmin database. Example 4-11 shows how we could use the ifx_grid_function procedure to add a new dbspace to all of the servers within the grid.

Example 4-11 Adding a new dbspace to the servers within the grid

```
database sysadmin;
EXECUTE FUNCTION ifx_grid_function('mygrid',
    'task("create dbspace", "dbsp2", "chk2", "1G,","0")',
    'myTag')
```

The syntax of the function is:

```
ifx_grid_function(<grid_name>,<function_statement>,<tag>)
```

Where the parameters have the following meanings:

- grid_name is the name of the grid on which the function is to be executed.
- function_statement is the text of the function be executed.
- tag is the optional tag used to monitor the progress of the statement.

As with the grid statement execution, any data that is changed by **ifx_grid_function** is not replicated by default.

4.2.12 Performing a grid procedure execution (ifx_grid_procedure)

In addition to the execution of a function, we can also execute a procedure as a grid operation, which might be useful if you have to execute a routine on all of the nodes, possibly to create data for a special report.

The format of this procedure is:

```
ifx_grid_procedure(<grid_name>,<procedure_statement>,<tag>)
```

Where the parameters have the following meanings:

- grid_name is the name of the grid on which the function is to be executed.
- procedure_statement is the text of the procedure to be executed.
- tag is the optional tag used to monitor the progress of the statement.

It is also possible that we create the procedure as a member of the grid and also execute it, as shown in Example 4-12.

Example 4-12 Using the ifx_grid_procedure

```
execute procedure ifx_grid_connect('mygrid', 1);
create table tab2(
    servname    char(10),
    mynum       integer);

create procedure insrow1(mynumparm integer)
    insert into tab2 (servname, mynum)
        values (DBSERVERNAME, mynumparm);
end procedure;

execute procedure ifx_grid_procedure('grid1', 'insrow1(1)');
```

If we issue “select * from tab2” on each of the servers at this point, we notice that the contents are not the same (Table 4-6). This is because by default the data changes of a grid operation are not replicated. Each server has only its own row. It does not have the rows that were inserted on the other nodes.

Table 4-6 Tab2 content

Server	servername	mynum
serv1	serv1	1
serv2	serv2	1
serv3	serv3	1
serv4	serv4	1

4.2.13 Propagating the results of a grid procedure execution

We saw in 4.2.12, “Performing a grid procedure execution (ifx_grid_procedure)” on page 84 that, by default, the results of **`ifx_grid_execute()`**, **`idx_grid_function()`**, and **`ifx_grid_procedure()`** are not replicated by ER. There are cases where this outcome is not the desired behavior. For example, if we had created a stored procedure to process data on each of the servers within a Flexible Grid with the idea that we could create a consolidated report, then we might expect that data to be replicated back to a central server for analysis.

It is possible to change the behavior so that the data is replicated even if the changes are part of a grid operation. This is done by the ifx_set_erstate() procedure, which can perform the following actions:

- Turn off ER snooping for this transaction:

  ```
  execute procedure ifx_set_erstate(0)
  ```

- Turn on ER snooping for this transaction:

  ```
  execute procedure ifx_set_erstate(1)
  ```

If we modify the prior SQL script shown in Example 4-13, we observe that the data is replicated as well.

Example 4-13 Propagating the result with a grid procedure

```
execute procedure ifx_grid_connect(‘grid1’, 1);
create table tab3(
    servname    char(10),
    mynum       integer
    );
```

```
create procedure insrow2(mynumparm integer)
    execute procedure ifx_set_erstate(1);
    insert into tab3 (servname, mynum)
        values (DBSERVERNAME, mynumparm);
end procedure;

execute procedure ifx_grid_procedure('grid1', 'insrow2(1)');

...(and then)...
select * from tab3;
servname        mynum
serv1               1
serv3               1
serv4               1
serv2               1
```

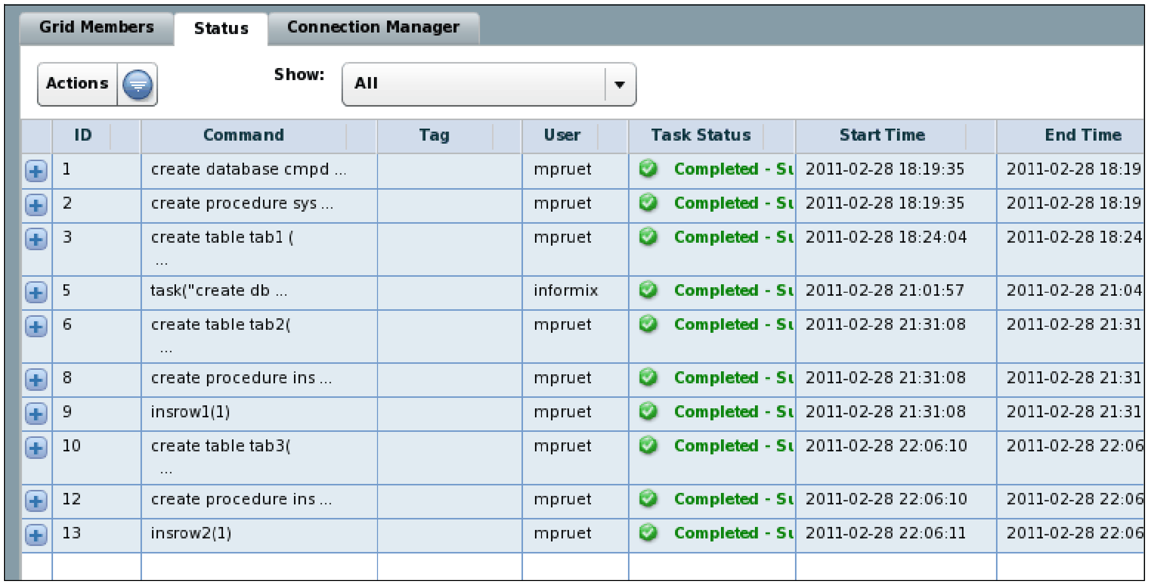
4.2.14 Examining the results of grid operations

You can check the results of the grid operations by using OAT or the cdr utility.

Using OAT to review the results

You can use OAT to review the results of grid operations. Navigate to the main Flexible Grid window by selecting **Replication** → **Grid**. Click the grid that you want to display and then click the **Status** tab.

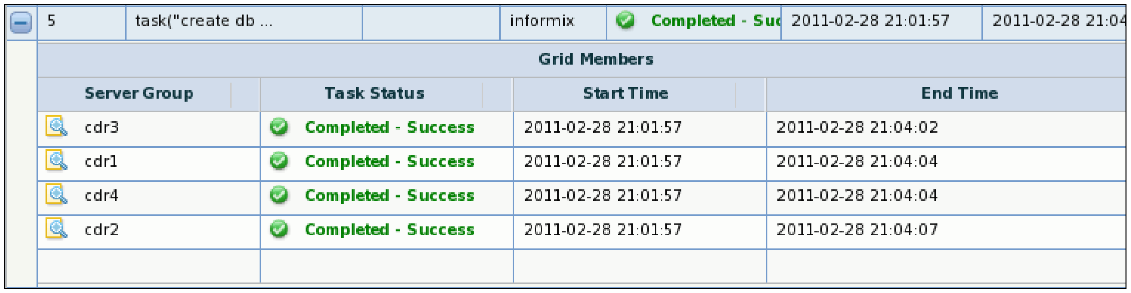
Figure 4-17 illustrates a sample view of grid operations.



	ID	Command	Tag	User	Task Status	Start Time	End Time
+	1	create database cmpd ...		mpruet	✓ Completed - Su	2011-02-28 18:19:35	2011-02-28 18:19
+	2	create procedure sys ...		mpruet	✓ Completed - Su	2011-02-28 18:19:35	2011-02-28 18:19
+	3	create table tab1 (...		mpruet	✓ Completed - Su	2011-02-28 18:24:04	2011-02-28 18:24
+	5	task("create db ...		informix	✓ Completed - Su	2011-02-28 21:01:57	2011-02-28 21:04
+	6	create table tab2(...		mpruet	✓ Completed - Su	2011-02-28 21:31:08	2011-02-28 21:31
+	8	create procedure ins ...		mpruet	✓ Completed - Su	2011-02-28 21:31:08	2011-02-28 21:31
+	9	insrow1(1)		mpruet	✓ Completed - Su	2011-02-28 21:31:08	2011-02-28 21:31
+	10	create table tab3(...		mpruet	✓ Completed - Su	2011-02-28 22:06:10	2011-02-28 22:06
+	12	create procedure ins ...		mpruet	✓ Completed - Su	2011-02-28 22:06:10	2011-02-28 22:06
+	13	insrow2(1)		mpruet	✓ Completed - Su	2011-02-28 22:06:11	2011-02-28 22:06

Figure 4-17 Viewing the results of grid operations

If you want to see the summary results from each of the nodes, click the ⊞ icon in the first column. Figure 4-18 shows the expansion of command 5.



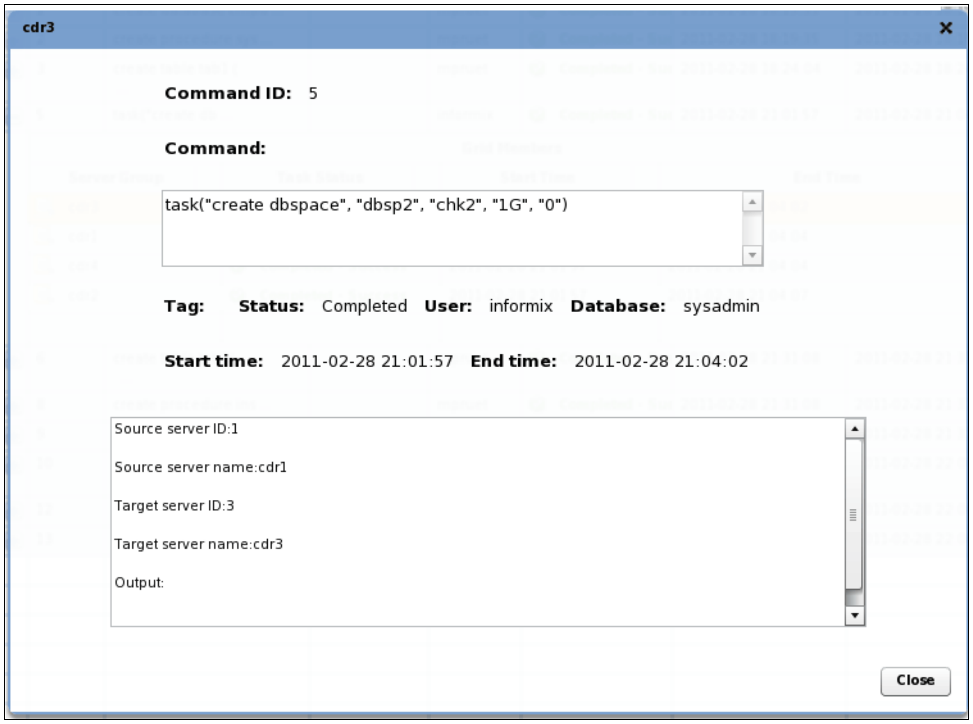
−	5	task("create db ...		informix	✓ Completed - Suc	2011-02-28 21:01:57	2011-02-28 21:04

Grid Members

Server Group	Task Status	Start Time	End Time
cdr3	✓ Completed - Success	2011-02-28 21:01:57	2011-02-28 21:04:02
cdr1	✓ Completed - Success	2011-02-28 21:01:57	2011-02-28 21:04:04
cdr4	✓ Completed - Success	2011-02-28 21:01:57	2011-02-28 21:04:04
cdr2	✓ Completed - Success	2011-02-28 21:01:57	2011-02-28 21:04:07

Figure 4-18 Execution on each of the nodes

If you want to see the detail results for the grid execution on a single node, click the icon in the Server Group column. Figure 4-19 shows an example.


cdr3

Command ID: 5

Command:

```
task("create dbspace", "dbsp2", "chk2", "1G", "0")
```

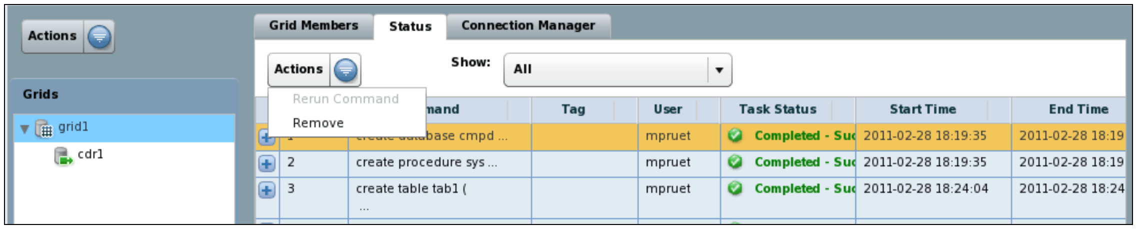
Tag: **Status:** Completed **User:** informix **Database:** sysadmin

Start time: 2011-02-28 21:01:57 **End time:** 2011-02-28 21:04:02

```
Source server ID:1

Source server name:cdr1

Target server ID:3

Target server name:cdr3

Output:
```

Figure 4-19 Detailed information from the execution of a grid command on a single node

The results of commands are retained until they are specifically removed. This action is done by highlighting a command and then clicking the **Action** button. One of the actions is Remove (see Figure 4-20). Click **Remove** in the **Actions** drop-down menu, and you see a confirmation window. After you confirm the removal, the row is deleted.



	mand	Tag	User	Task Status	Start Time	End Time
1	create database cmpd ...		mpruet	Completed - Suc	2011-02-28 18:19:35	2011-02-28 18:19
2	create procedure sys ...		mpruet	Completed - Suc	2011-02-28 18:19:35	2011-02-28 18:19
3	create table tab1 (...		mpruet	Completed - Suc	2011-02-28 18:24:04	2011-02-28 18:24

Figure 4-20 Removing a the grid command status

Using cdr to view the status of grid commands

You can also use the **cdr list grid** command to view the results of grid commands. There are several filters that can be used to restrict the output of the command. These filters can be seen by executing **cdr list grid -h**, as shown in Example 4-14.

Example 4-14 cdr list grid -h

```
cdr list grid -h
usage: cdr list grid [-c server] -cCSsvnax [-t tagname] [ GridName ]
 -c server --connect=server  connect to server
 -C cmd    --command=<command ID>
 -t        --tag=<tag name>
 -S source --source=<source name>
 -s        --summary
 -v        --verbose
 -n        --nacks
 -a        --acks
 -p        --pending
```

If you simply execute **cdr list grid -v**, you see the status of each grid operation from each server as well as the original operation.

Example 4-15 cdr list grid -v

```
cdr list grid -v
...
Node:cdr1 Stmtid:10 User:mpruet Database:cmpdb 2011-02-28 22:06:10
create table tab3(
    servname    char(10),
    mynum       integer
    )
ACK cdr1 2011-02-28 22:06:10
ACK cdr2 2011-02-28 22:06:11
ACK cdr3 2011-02-28 22:06:10
ACK cdr4 2011-02-28 22:06:10

Node:cdr1 Stmtid:11 User:informix Database:syscdr 2011-02-28 22:06:10
Define Repl G65540_1_11_tab3 for cmpdb:mpruet.tab3
ACK cdr1 2011-02-28 22:06:10
ACK cdr2 2011-02-28 22:06:11
ACK cdr3 2011-02-28 22:06:10
ACK cdr4 2011-02-28 22:06:11

Node:cdr1 Stmtid:12 User:mpruet Database:cmpdb 2011-02-28 22:06:10
create procedure insrow2(mynumparm integer)
execute procedure ifx_set_erstate(1);
insert into tab3 (servname, mynum)
```

```
    values (DBSERVERNAME, mynumparm);
end procedure;
ACK cdr1 2011-02-28 22:06:10
ACK cdr2 2011-02-28 22:06:11
ACK cdr3 2011-02-28 22:06:10
ACK cdr4 2011-02-28 22:06:11
```

Viewing the output of grid functions

The only difference between a stored procedure and a stored function is that a function returns output. When you execute a grid function, the output of the function's execution is recorded in the status output. Informix stores the output of the function as an *lvarchar*, which is subsequently replicated back as part of the status. We use Example 4-16 as an example.

Example 4-16 Stored function returning values

```
execute procedure ifx_grid_connect(‘grid1’, 1);

ccreate function getid()
    returns varchar(128), varchar(128)

    define servname   varchar(128);
    define servnum    varchar(128);
    LET servname = DBSERVERNAME;
    select trim(t.cf_effective) into servnum
        from sysmaster:syscfgtab t
        where t.cf_name = "SERVERNUM";

    return servnum, servname;
end function;

execute function ifx_grid_function('grid1', 'getid()');
```

We can either examine the OAT panels to see the detail output or we can examine the output of **`cdr list grid -v`**. Example 4-17 shows the **`cdr list grid`** output.

Example 4-17 cdr list grid output

```
cdr list grid -v
Node:cdr1 Stmtid:20 User:mpruet Database:cmpdb 2011-03-01 11:45:00
getid()
ACK cdr1 2011-03-01 11:45:00
    '1','serv1'
ACK cdr2 2011-03-01 11:45:01
    '2','serv2'
ACK cdr3 2011-03-01 11:45:00
```

```
    '3','serv3'
ACK cdr4 2011-03-01 11:45:01
    '4','serv4'
```

Under each of the ACKs from each of the servers, there is an output line. This is a character string in comma delimited format of the output from that node. The output from the execution on the server cdr1 was `‘1’,’serv1’`, from cdr2 is `‘2’,’serv2’`, and so on.

4.3 Administration tasks

As a DBA, you always have administrative tasks to be performed. It is much easier to perform administrative tasks using Flexible Grid compared to ER, where each task has to be executed independently on each server.

The administrative tasks for Flexible Grid including the following:

- Adding a node to an existing grid
- Removing a node from a grid
- Enabling a node as the source of grid operations
- Disabling a node as a source of grid operations
- Authorizing a user to perform grid operations
- Removing authorization from a user to perform grid operations
- Adding a space on all the servers in the grid
- Changing configuration parameter setting
- Running Update Statistics

In the previous sections of this chapter, we discussed certain tasks regarding the administration of the grid. Here we describe examples of the administrative tasks that can be run using SQL administration API commands across all servers in the grid.

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 across all the servers in the grid:

- Run the ifx_grid_function() function with the SQL administration API command as the second argument.
- 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 because the SQL admin() returns an integer value indicating success or failure instead of SQLCODE. When **ifx_grid_function()** is run, the function is executed but the results are not processed.

Here are two grid administrative task examples, one where the SQL admin API passed and the other where it failed:

- Changing a configuration parameter setting

  Example 4-18 shows how to change the location of the **onconfig** parameter, for example, LOG_STAGING_DIR, and the result from **cdr list grid**, which shows success.

  Example 4-18 Changing a configuration parameter for the grid

```
DATABASE sysadmin;
EXECUTE FUNCTION ifx_grid_function('grid_1', 'admin("set onconfig
permanent", "LOG_STAGING_DIR","/work/informix/stg_log")');

xmach6:/gpfs1/informix>cdr list grid --acks grid_1

Node:newyork Stmtid:40 User:informix Database:sysadmin 2011-03-03 14:26:12
admin("set onconfig permanent", "LOG_STAGING_DIR","/work/informix/stg_log")
ACK frankfurt 2011-03-03 14:26:14
    '106'
ACK london 2011-03-03 14:26:14
    '106'
ACK newyork 2011-03-03 14:26:14
    '193'
ACK singapore 2011-03-03 14:26:14
    '106'
```

  Note: The following example must be run in the sysadmin database.

- Adding a dbspace in the grid

  Example 4-19 shows how to add a dbspaces in the grid and the result from **`cdr list grid`**, which shows failure in execution in one of the nodes.

  Example 4-19 Adding a dbspace in the grid

```
task("create dbspace","dbs5", "/usr2/stress/grid/dbs5","2G","0")
ACK frankfurt 2011-03-03 17:43:37
    'Space 'dbs5' added.'
ACK london 2011-03-03 17:43:05
    'Unable to create file /usr2/stress/grid/dbs5'
ACK newyork 2011-03-03 17:43:37
    'Space 'dbs5' added.'
ACK singapore 2011-03-03 17:43:52
    'Space 'dbs5' added.'
```

4.4 Error recovery

Unlike Enterprise Replication, Flexible Grid propagates SQL statements but does not perform conflict resolution. There may be times when you might run into conflicts such as:

- A table already exists on one of the nodes.
- TABLE is locked.
- Space does not exist.
- An alter table fails because the table is locked.

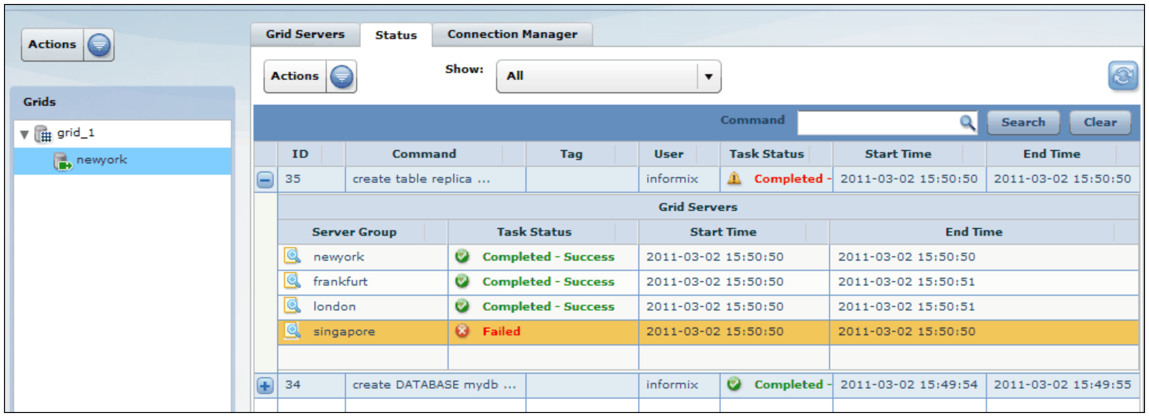
Finding the error and performing error recovery is fairly easy in Flexible Grid. Using the OAT interface, you can review the results of the grid operation and re-run the failed command after examining the failures.

As an example, we create the dbs1 dbspace in all of the nodes and the dbs3 dbspace in all nodes except for the singapore node. We then create a table in the grid called replicated_tab with fragment by expression in dbs1 and dbs3. This operation fails on execution of the singapore node, as the dbs3 dbspace does not exist. On taking corrective action, which is to add the dbspace on the singapore node, we can then re-run the create table statement on the failed node.

In the following sections, we show how to review a failed grid operation and take appropriate action.

4.4.1 Using OAT to review the results

To view the results using OAT, select **Replication** → **Grid** → **Status**. Figure 4-21 shows the status of the grid operation.



Grid Servers | Status | Connection Manager

Actions | Show: All

Grids
- grid_1
  - newyork

Command | Search | Clear

	ID	Command	Tag	User	Task Status	Start Time	End Time
−	35	create table replica ...		informix	⚠ Completed -	2011-03-02 15:50:50	2011-03-02 15:50:50

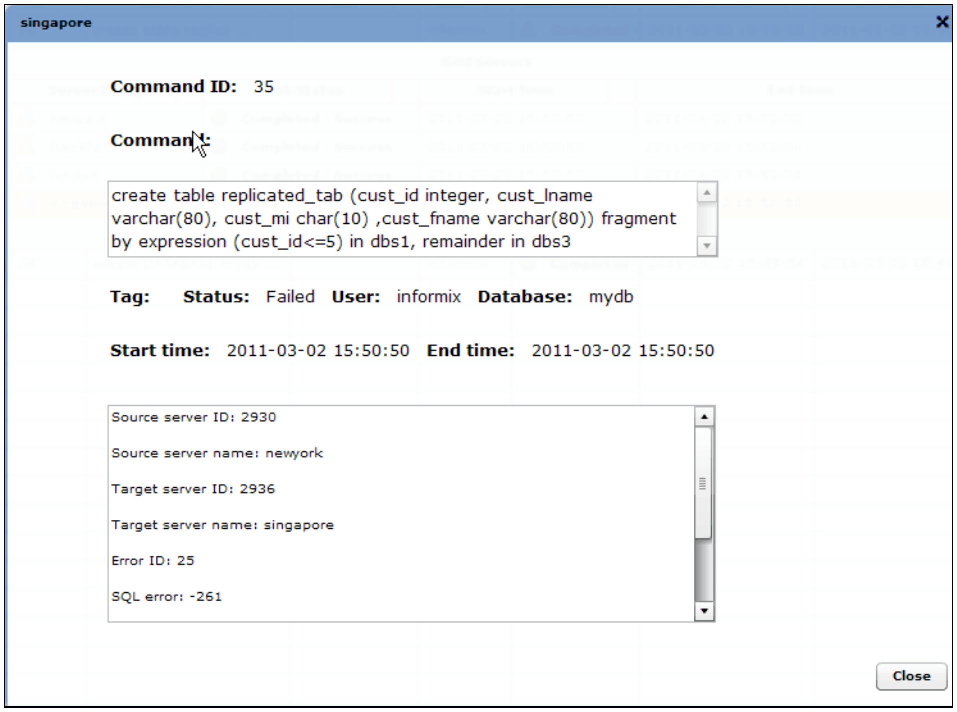
Grid Servers

Server Group	Task Status	Start Time	End Time
newyork	✓ Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:50
frankfurt	✓ Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
london	✓ Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
singapore	✗ Failed	2011-03-02 15:50:50	2011-03-02 15:50:50

	ID	Command	Tag	User	Task Status	Start Time	End Time
+	34	create DATABASE mydb ...		informix	✓ Completed -	2011-03-02 15:49:54	2011-03-02 15:49:55

Figure 4-21 View results of the grid operation

To discover the details of the failure, click the error. Figure 4-22 shows the detailed message.



Command ID: 35

Command:

```
create table replicated_tab (cust_id integer, cust_lname
varchar(80), cust_mi char(10) ,cust_fname varchar(80)) fragment
by expression (cust_id<=5) in dbs1, remainder in dbs3
```

Tag: **Status:** Failed **User:** informix **Database:** mydb

Start time: 2011-03-02 15:50:50 **End time:** 2011-03-02 15:50:50

```
Source server ID: 2930

Source server name: newyork

Target server ID: 2936

Target server name: singapore

Error ID: 25

SQL error: -261
```

Figure 4-22 Viewing the grid operation error details

The detailed error messages show that the create table operation resulted in SQL code -261 or ISAM error -130, which translates to table creation failed due to a non-existent dbspace.

You can also run the **cdr list grid** command with the --nacks option to see the details. Example 4-20 shows the non-acknowledge statement in the grid.

Example 4-20 Viewing the grid operation result using the cdr utility

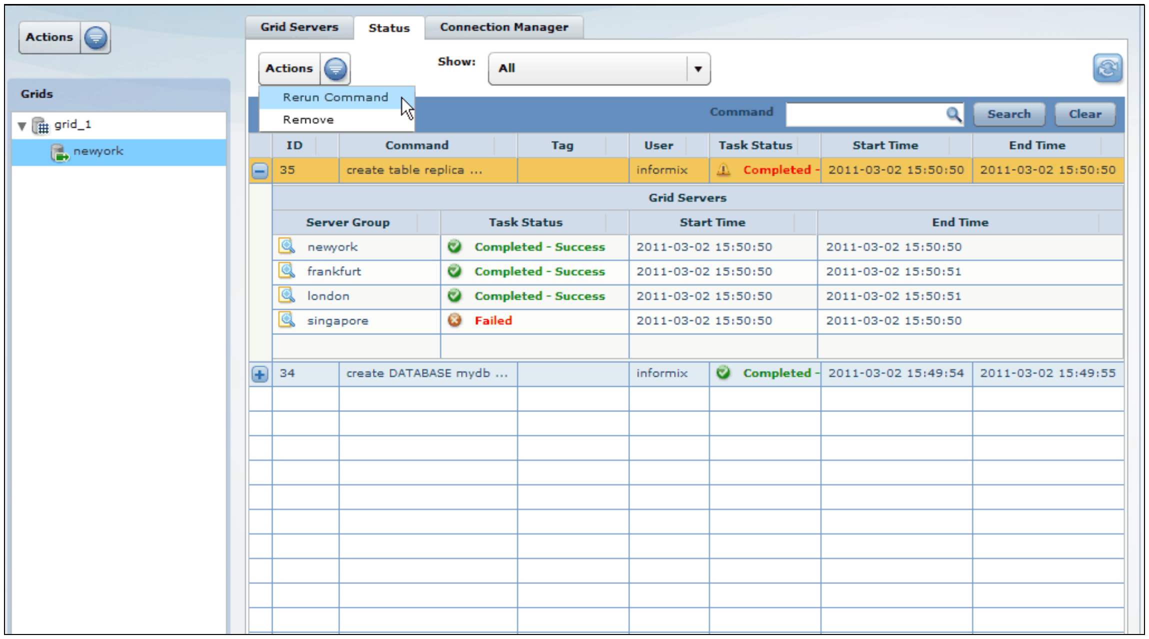
```
xmach6:/gpfs1/informix>cdr list grid --nacks grid_1
Grid                     Node                     User
------------------------ ------------------------ ------------------------
grid_1                   frankfurt
                         london
                         newyork*                 informix
                         singapore
Details for grid grid_1

Node:newyork Stmtid:35 User:informix Database:mydb 2011-03-02 17:50:50
```

```
create table replicated_tab (cust_id integer, cust_lname varchar(80), cust_mi
char(10) ,cust_fname varchar(80)) fragment by expression (cust_id<=5) in dbs1,
remainder in dbs3
NACK singapore 2011-03-02 17:50:50 SQLERR:-261 ISAMERR:-130
     Grid Apply Transaction Failure

Node:newyork Stmtid:36 User:informix Database:syscdr 2011-03-02 17:50:50
Define Repl grid_192020502_2930_36 for mydb:informix.replicated_tab
NACK singapore 2011-03-02 17:50:50 SQLERR:0 ISAMERR:0
     Grid Apply Transaction Failure
```

Rerunning the failed grid commands

We can correct this problem by adding the dbs3 dbspace on the singapore node and then rerunning the failed statement.

To rerun a command using OAT, open the Status tab, select the command to be rerun, then select **Rerun Command** from the **Actions** drop-down menu, as shown in Figure 4-23.



ID	Command	Tag	User	Task Status	Start Time	End Time
35	create table replica ...		informix	Completed -	2011-03-02 15:50:50	2011-03-02 15:50:50

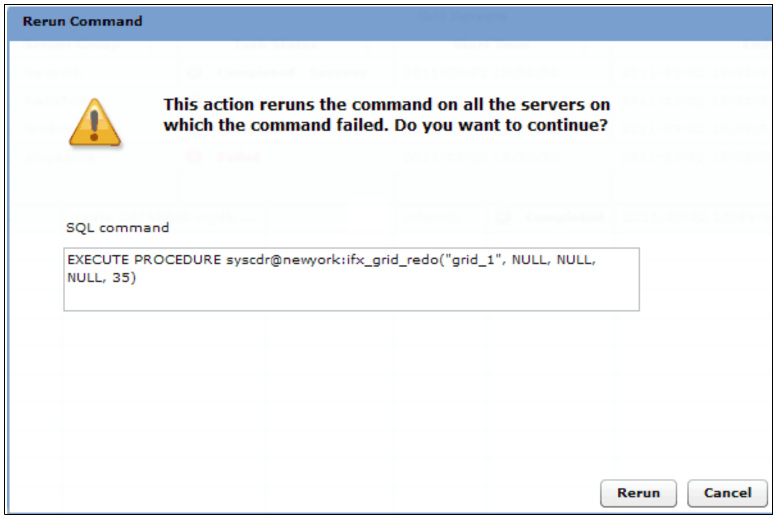
Grid Servers

Server Group	Task Status	Start Time	End Time
newyork	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:50
frankfurt	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
london	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
singapore	Failed	2011-03-02 15:50:50	2011-03-02 15:50:50

ID	Command	Tag	User	Task Status	Start Time	End Time
34	create DATABASE mydb ...		informix	Completed -	2011-03-02 15:49:54	2011-03-02 15:49:55

Figure 4-23 Rerun a failed command

OAT asks you to confirm the rerun command action, as shown in Figure 4-24.



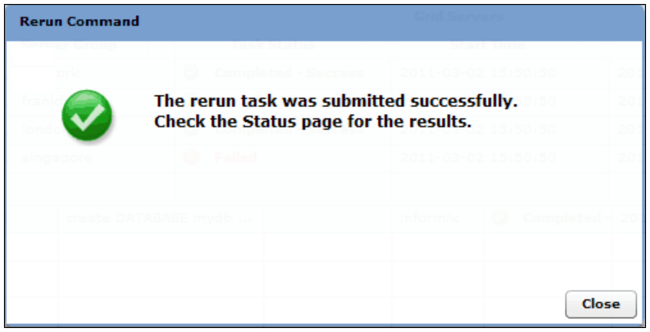
This action reruns the command on all the servers on which the command failed. Do you want to continue?

SQL command

```
EXECUTE PROCEDURE syscdr@newyork:ifx_grid_redo("grid_1", NULL, NULL,
NULL, 35)
```

Figure 4-24 Rerun command confirmation

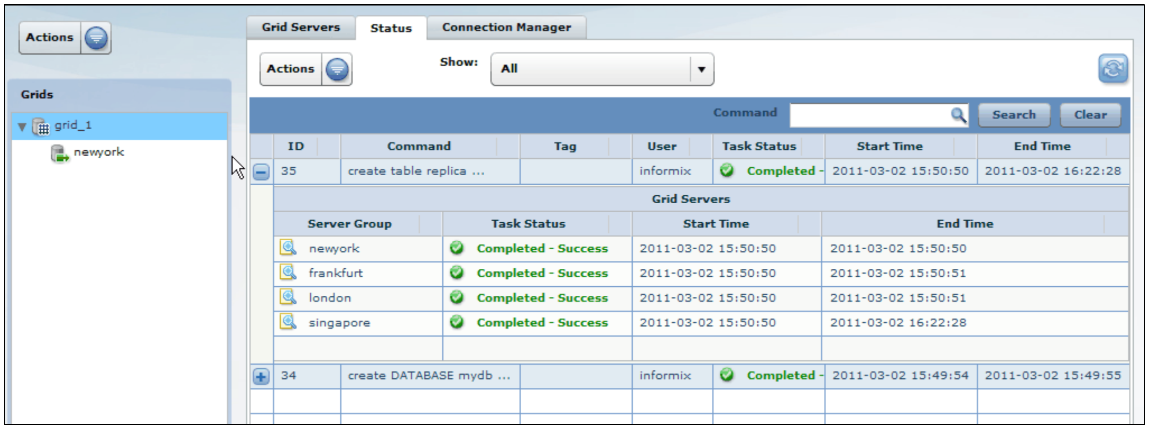
Figure 4-25 shows that the create table command ran successfully.



The rerun task was submitted successfully. Check the Status page for the results.

Figure 4-25 Command reran successfully

When we check the status again, there is no error, as shown in Figure 4-26.



ID	Command	Tag	User	Task Status	Start Time	End Time
35	create table replica ...		informix	Completed -	2011-03-02 15:50:50	2011-03-02 16:22:28

Grid Servers

Server Group	Task Status	Start Time	End Time
newyork	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:50
frankfurt	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
london	Completed - Success	2011-03-02 15:50:50	2011-03-02 15:50:51
singapore	Completed - Success	2011-03-02 15:50:50	2011-03-02 16:22:28

ID	Command	Tag	User	Task Status	Start Time	End Time
34	create DATABASE mydb ...		informix	Completed -	2011-03-02 15:49:54	2011-03-02 15:49:55

Figure 4-26 Grid operation completes successfully

4.4.2 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.

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.

The syntax of the ifx_grid_redo() procedure is:

```
EXECUTE PROCEDURE ifx_grid_redo (<grid_name>, <source_server>, <target_server>,
<tag>, <command_id>)
```

Where the meaning of the procedure parameters are:

- grid _name: This is the name of the grid.
- source_server: This is the replication server from which the command was run. This parameter can be NULL.
- target_server: This is the replication server on which to rerun the routine. This parameter can be NULL.
- tag: This is a character string identifying the operation to be rerun. This parameter can be NULL.
- command_id: This is the ID number of the command to be rerun on the grid. This parameter can be NULL, one ID, or multiple IDs.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

The following example reruns failed commands on every server in the grid on which those commands failed:

```
EXECUTE PROCEDURE ifx_grid_redo('grid_1');
```

The following example reruns the command with the ID of 35 that originated on newyork server and must be rerun on the singapore server:

```
EXECUTE PROCEDURE ifx_grid_redo('grid_1', 'newyork', 'singapore', NULL, '35');
```

The following example reruns all commands that failed on the singapore server:

```
EXECUTE PROCEDURE ifx_grid_redo('grid_1', NULL, 'singapore');
```

5

Administering Enterprise Replication

In this world of on demand information, more and more companies find themselves managing distributed databases that spread corporate information to all corners of the enterprise. Data replication can be the key to help ensure an effective and consistent delivery of corporate data enterprise-wide. IBM Informix provides Enterprise Replication, an embedded high performance solution for distributing, sharing, and maximizing the availability of corporate information.

Replication systems must provide high performance without burdening the source database and without requiring application changes. They have to maximize data availability and ensure consistent delivery of the data. In addition, database administrators must be able to configure and manage all the components of the replication systems in a manner that uses the enterprise computing resources most effectively.

In this chapter, we introduce certain basic use cases, concepts, and terminology of Enterprise Replication. *IBM Informix Enterprise Replication* Guide, SC27-3539 is the authoritative source for information about Enterprise Replication. It has all the details of the commands and allowed configurations. Take time to study that manual carefully in addition to the material presented here.

5.1 A brief overview

Enterprise replication can be simply defined as the process of propagating a defined and configured replicate set from one database server to another. Database replication is important because it enables enterprises to provide users with access to current data where and when they need it. It can also provide a wide spectrum of benefits, including improved performance when centralized resources get overloaded, increased data availability, capacity relief, and support for data warehousing to facilitate decision support.

5.1.1 Why replicate data

One key consideration in designing an efficient replication system is a clear understanding of the business requirements. An enterprise typically has a number of different requirements for using data replication. Some organizations might want to increase fault tolerance, thereby duplicating the critical information in more than one location; some might want to increase data availability and performance of their application by using local replicates to avoid WAN network traffic.

Here are more examples of the business requirements for which organizations might want to replicate data:

- A global enterprise wants to provide 24x7 customer support using call centers that are distributed in various locations around the world. To meet this business requirement, you need the ability to locally update records and then replicate them to the other call centers allowing them to function autonomously, even when original call centers are not available.
- A bank headquarters might need to send updated interest rates to its remote branches on a daily basis. In this case, data is updated in a central location (the headquarters) and distributed to multiple, read-only sites (branch offices).
- A retail store chain gathers point-of-sale information throughout the day that is then transmitted, at the end of the day, to the headquarters, where it is consolidated into the central data warehouse system to be used in various business analytics processes to determine current and future consumer trends.
- In an insurance claims processing organization, the adjuster might enter the damage claim on one system, and it might be reviewed or adjudicated by a reviewer on a different system using a separate database. In this case, the different parts of a workflow are processed on separate systems. The results of the work on each system are replicated to the system on which the next step will be performed.

There may be several other requirements where companies would want to replicate data, but providing data availability to the users and applications is usually one of the main requirements.

5.1.2 Why use Enterprise Replication

IBM Informix Enterprise Replication (ER) offers robust, easy-to-use, and flexible capabilities for propagating data changes from one database server to another. The flexible architecture of ER allows organizations to define their replication environments based on their specific business requirements. For example, you may choose to replicate entire databases or a few columns or rows of a table, and when and where to replicate them. You might want to replicate data from one database server to one or more database servers, or you might want to replicate data among a set of database servers where any of them may update the data.

ER is an asynchronous, log based data replication tool. Asynchronous replication is often preferred over synchronous replication because it can tolerate system and network failures. The Informix database server multithreaded architecture allows ER to perform its operations in parallel with other database activities, making it a high performance solution. ER does not compete with user transactions to capture changed data for replication; rather, data is collected from logical logs. An advantage of log based data replication is that it can be used with heterogeneous operating systems and multiple versions of Informix. For example, Informix 11.70 on Linux can replicate data with Informix 11.50 on AIX.

Enterprise Replication provides a rich set of functionality that facilitate DBAs to perform various administration tasks easily, such as:

- Setting up replication over a large number of database servers.
- Adding a new database server into an existing replication environment and performing initial data synchronization while replication is active.
- Updating the schema of the replicated tables without interrupting the active data replication.
- Managing the replication domain from a single point of control, either by using the web browser based OpenAdmin Tool (OAT), or the cdr command-line utility.
- Performing data consistency checks and re-synchronization to correct data mismatches between replicated tables, if for some reason data is not in sync.
- Changing the replication configuration dynamically. Any changes made to the ER configuration are automatically propagated to other servers.
- Encrypted data transmissions between database servers.

Enterprise Replication is best suited for embedded applications due to the fact that administration tasks can be automated using SQL based interface and event alarms that are generated during the replication process.

Enterprise replication integrates well with a high availability (HA) cluster. Any of the database servers in an ER domain may be part of a HA cluster. Using an HA cluster with ER helps ensure that key nodes in the replication scheme are highly available so the flow of data is not interrupted.

5.2 An example replication scenario

In this chapter, we refer to the sample replication infrastructure introduced in Chapter 3, “Configuring a server for Flexible Grid and Enterprise Replication environments” on page 31 to illustrate some of the definitions and settings that ER provides with regard to the replicates. We provide detailed examples of those definitions and settings using both OAT and the cdr utility.

The database that is used for the examples in this chapter is called *stores9* and was created by the dbaccessdemo9 utility. We discuss all the options for defining the data flow in a replicate in simple examples based on the *customer* table.

Here are the steps involved in setting up an existing infrastructure to start replicating data across servers using Enterprise Replication:

- Preparing the database server
- Preparing the data
- Defining the replicates
- Starting Enterprise Replication

5.3 Preparing the database server

After you decide on the topology, you need to prepare the database servers that are to be used in the replication before starting replication setup. See Chapter 3, “Configuring a server for Flexible Grid and Enterprise Replication environments” on page 31 for details about how to prepare the database servers in an existing infrastructure to participate in Enterprise Replication.

5.4 Preparing the data

Before setting up the ER components for the servers and the data flow, first verify that the prerequisites are met. If they are not, the definition of the replicates simply fail. In this section, we briefly discuss some essential prerequisites that must be applied to the existing data.

5.4.1 Logged tables

The evaluation and distribution of the data in the ER environment defined by the replicates is based on the logical log. All base tables on which replicates are defined have to be standard logged tables and have to belong to a database with logging defined.

The **`cdr define replicate`** command, in an attempt to create a replicate on a non-logged base table, returns an error, as shown in Example 5-1.

Example 5-1 cdr define replicate on a table in a non-logged database

```
cdr define replicate -M newyork -C ignore -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
command failed -- database not logged  (7)
```

Raw tables and temporary tables are not allowed in the definition of a replicate. In addition, raw tables do not allow the creation of a primary key constraint. The error message returned by the **`cdr define replicate`** command, in the attempt to create the replicate based on a raw table, is the same as for a missing primary key, as shown in Example 5-2 on page 106.

5.4.2 Primary keys

Primary keys (PKs) are required in any table that is used in replication. The primary key is necessary so that the receiving DBMS can accurately identify which rows are involved in each transaction. In addition, a comparison of the primary key column in each replicated change is the way conflicts are found. If the table does not define a primary key, the **`cdr define replicate`** command returns an error, as shown in Example 5-2.

Example 5-2 Missing primary key on a base table for cdr define replicate

```
cdr define replicate -C ignore -S row -A -R raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'
stores9@london:informix.customer 'select * from informix.customer'

#output
command failed -- table does not contain primary key  (18)
```

If any of the tables that must be replicated do not have a primary key, the table may be altered to have an artificial primary key or ERKEY shadow columns.

Preparing a table with an artificial primary key

Altering a table to have an artificial primary key can be done in at least two ways:

- The first method is to define a composite primary key that is some set of existing columns. This technique avoids changing the row size and storage requirements of the table.
- A second way to define an artificial primary key is to add a column. The new column might be of the SERIAL data type or an integer that will be populated from a sequence. In either case, the new column must be added and it must be populated with data.

A common source of conflicts in an update anywhere environment is the use of the SERIAL data type in the primary key.

In our example environment, we want to replicate the customer table across the newyork, singapore, and frankfurt servers in an update-anywhere scenario, with a conflict resolution set to ignore. Each of the servers have a specific number of user sessions inserting new customers into the table that are sent out to the other server. Because each of the tables on the different replication servers maintain the same values for the serial with no separate ranges for the others, there will be collisions during the data distribution, because the same primary key already exists on the target server.

To minimize these conflicts, the CDR_SERIAL configuration parameter may be used. CDR_SERIAL defines for a replication server the common serial data type behavior for tables with serials defined in a replicate. You have to choose a setting for each replication server in the infrastructure, where there is no overlapping in the value range. A common sample setting is a configuration with 10,1 for the first server, 10,2 for the next one, and so on. Finally, you can achieve something like a sequences behavior across the replication servers.

Preparing a table without primary keys

Preparing a table without a primary key for enabling replication can be done by adding the ERKEY shadow columns to the table. Complete the following steps:

1. Add the ERKEY shadow columns when creating the table or alter the table to add the ERKEY shadow columns, if the table already exists.
2. Include the --erkey option when defining the replicate.

Example 5-3 shows how to create and alter the table to add ERKEY.

Example 5-3 Create and alter table with ERKEY

```
#create table
create table "informix".customer
  (
    customer_code  integer,
    customer_name  char(31),
    company_name   char(20)
) with ERKEY;

#OR

# alter table
alter table customer add ERKEY;

#define replicate with --erkey option
cdr define replicate --erkey -C ignore -S row -A -R raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'
stores9@london:informix.customer 'select * from informix.customer'
```

5.4.3 Data consistency across the replication server

At the time of the definition and the start of replicating the data across the replication servers, where the replicates are defined should be consistent. Make sure that all the data sets that are covered by the definition of the new replicates are the same on all servers to avoid the creation of the aborted transaction spooling (ATS) and row information spooling (RIS) files by the target database server. A manual investigation of the changes taken from the spooled ATS and RIS files can result in significant administrative impact to get the tables in sync after the reported errors.

To make this more clear, we define a simple example situation. We plan to replicate the customer table between the newyork, singapore, and frankfurt servers. The table does exists, but we have only data loaded on frankfurt and newyork. After the definition of the replicate, we attempt to delete one of the rows. This deletion will be successful on newyork and frankfurt, but on singapore, the server will report ATS and RIS files, as shown in Example 5-4.

Example 5-4 Avoid data inconsistencies at the replicate startup time

```
#On newyork, frankfurt and singapore
dbaccess -e stores9 << EOF
create table "informix".customer
  (
    customer_code  integer,
    customer_name  char(31),
    company_name   char(20),
    PRIMARY KEY (customer_code)
  ) with crcols;
EOF

#On frankfurt and newyork
dbaccess -e stores9 << EOF
load from customer insert into customer;
EOF

#define the replicate
cdr define replicate -M newyork -u -C ignore -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#delete a row on newyork
dbaccess -e stores9 << EOF
delete from customer where customer_code=110;
EOF
```

```
#logfile on singapore
13:07:26  CDR CDRD_6: transaction aborted (All rows in a transaction defined
with row scope were rejected) with sql error 0 isam error 0.
13:07:26  CDR CDRD_6: failed rows spooled to file
/files/IDS/RIS/ris.singapore.newyork.D_6.110223_13:07:26.1
13:07:26  CDR CDRD_6: failed transaction spooled to file
/files/IDS/ATS/ats.singapore.newyork.D_6.110223_13:07:26.2

#RIS file on singapore
TXH Source ID:4930 / Name:newyork / CommitTime:11-02-23 13:07:26
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-02-23 13:07:26
----------
RRH Row:1 / Replicate Id: 323092524 / Table: stores9@informix.customer /
DbOp:Delete
RRH CDR:7 (Error: Delete aborted, row does not exist in target table) / SQL:0 /
ISAM:0
RRD 110|Roy Jaeger|AA Athletics
==========
TXH Transaction aborted
TXH ATS file:/files/IDS/ATS/ats.singapore.newyork.D_6.110223_13:07:26.2 has
also been created for this transaction

#ATS file on singapore
TXH RIS file:/files/IDS/RIS/ris.singapore.newyork.D_6.110223_13:07:26.1 has
also been created for this transaction
==========
TXH Source ID:4930 / Name:newyork / CommitTime:11-02-23 13:07:26
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-02-23 13:07:26
TXH Number of rows processed when transaction was aborted:1
TXH All rows in a transaction defined with row scope were rejected
TXH CDR:7 (Error: Delete aborted, row does not exist in target table) / SQL:0 /
ISAM:0
----------
RRH Row:1 / Replicate Id: 323092524 / Table: stores9@informix.customer /
DbOp:Delete
RRD 110|Roy Jaeger|AA Athletics
```

5.4.4 Shadow columns

In an update-anywhere scenario, you have to enable conflict resolution rules for the replicates. In our example environment, there are some updates on the same row on the newyork server that can be done at the same time on the frankfurt server. All the changes are replicated to the other servers. Assume that on the singapore server both updates arrive and that no conflict resolution definition exists. It is then a question of the arrival time of the changed row that determines which row version is finally seen on this database server.

ER defines several strategies for conflict resolution, which are discussed in “Conflict resolution policy” on page 132. To prepare the table to be used within an update-anywhere replicate and to support conflict resolution based on time stamps or on stored procedures, the table has to be created with shadow columns. If an existing table is added into the replication, the table has to be altered to add the shadow columns. Refer to Example 5-5 for the appropriate SQL statements.

Example 5-5 Create and alter table with crcols

```
#create table
create table "informix".customer
  (
    customer_code  integer,
    customer_name  char(31),
    company_name   char(20),
    PRIMARY KEY (customer_code)
  ) with crcols;

# alter table
alter table customer add crcols;
```

With shadow columns, we refer to two columns in the table named *cdrserver* and *cdrtime*. They are not visible in dbaccess. If you use **oncheck**, you might only notice a difference in the rowsize after adding the columns to the table. Currently, the shadow columns advance the rowsize of an existing table by 8.

Shadow columns and typed tables

To prepare a typed table for time stamp based conflict resolution, you have to unload, drop, and recreate the table with the crcols. Typed tables do not support the alter statement to add the crcols to an existing table. Such an operation will fail with an error -9802.

To show the preparation of a typed table, we use the customer table from the superstores_demo database created by the dbaccessdemo_ud utility. The appropriate steps are shown in Example 5-6.

Example 5-6 Apply crcols to a typed table

```
dbaccess << EOF
database superstores_demo;

drop table whol_customer;
delete from customer_log;
unload to "/tmp/customer" select * from customer;
create table "informix".whol_customer of type 'informix'.customer_t
  (
    check (customer_type IN ('R' ,'W' )),
    check (credit_status IN ('D' ,'L' ,'R' ,'P' ,'N' )),
    primary key (customer_num)
  ) with crcols;
revoke all on "informix".customer from "public";

alter table "informix".whol_customer add constraint (foreign key (customer_loc)
references "informix".location );

create trigger "informix".whol_cust_trig update of credit_status on
    "informix".whol_customer referencing old as pre_upd new as post_upd

    for each row
        (
        insert into "informix".customer_log (cust_num,cust_name,
        username,update_time,old_status,new_status)  values
        (pre_upd.customer_num
        ,pre_upd.customer_name ,USER ,CURRENT year to fraction(3)
        ,pre_upd.credit_status ,post_upd.credit_status )
        );

load from "/tmp/customer" insert into whol_customer;

EOF
```

5.4.5 Delete tables

ER creates a delete table for each table defined in an update-anywhere scenario with a time stamp or a stored procedure based conflict resolution. This table will be automatically created at the time a new replicate based on this table is created. The database server stores in this table all deleted rows from the original table for a certain time frame. The schema of the table follows the schema of the original table.

The delete table is in addition to the shadow columns, because the time stamp comparison for deleted rows cannot work because they no longer exist in the original table. The delete table provides two major functions for the ER system. First is that after the deletion, the application is no longer able to see the row. Second, the row is not really gone; it remains for a certain time to ensure the capability of a time stamp comparison with later incoming changes to prevent inconsistencies.

5.4.6 Schema changes versus data changes

ER supports only the distribution of changes on data such as inserts, updates, and deletes. Any changes to the schema, such as create index, alter table, or truncate table, will not be replicated to the target, as shown in Example 5-7.

Example 5-7 Truncate table behavior on both sides

```
#create the database
dbaccess -e stores9 << EOF
create table "informix".customer
  (
    customer_code  integer,
    customer_name  char(31),
    company_name   char(20),
    PRIMARY KEY (customer_code)
  ) with crcols;

load from customer insert into customer;
EOF


#define the replicate -- make sure its a mastered replicate
cdr define replicate -M newyork -n n -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

#truncate the table on the master (newyork)
dbaccess -e stores9 << EOF
truncate table customer;
EOF

#verification on the master side (newyork)
dbaccess -e stores9 << EOF
select count(*) from customer;
EOF

#output
```

```
(count(*))

0

#verification on the target side ( singapore )
dbaccess -e stores9 << EOF
select count(*) from customer;
EOF

#output
(count(*))

28
```

5.4.7 User defined types

ER does support user defined types (UDTs) in the base tables. There are no changes required to apply a table with UDT to a replicate. This includes user defined row types, as we have already shown in Example 5-6 on page 111, and also extended data types, such as lvarchar, as shown in Example 5-8.

Example 5-8 lvarchar in a base table for a replicate

```
dbaccess -e stores9 << EOF
create table "informix".customer
  (
    customer_code  integer,
    customer_name  lvarchar(31),
    company_name   char(20),
    PRIMARY KEY (customer_code)
  ) with crcols;

load from customer insert into customer;
EOF

#define the replicate
cdr define replicate -M newyork -n n -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#insert a row on the source
dbaccess -e stores9 << EOF
insert into customer values (129, "Bill Smith", "Sports Center");
EOF
```

```
#verification on the target
dbaccess -e stores9 << EOF
select * from customer where customer_code=129;
EOF

#output
customer_code  129
customer_name  Bill Smith
company_name   Sports Center
```

5.5 Defining the replicates

The definition of the replicates, replicate sets, or templates are based on the data flow required for the business need. In this section, we discuss how to define the replicates using OAT and the cdr utility.

5.5.1 Planning the replicates details

To define the replicates, you need to decide on the following items based on the data flow requirements:

- Replicate types
- Data flow direction
- Conflict resolution policy
- Column to be replicated
- Where to replicate the data

Replicate types

Replicating types related to data consistency. ER allows the definition of three different types of replicates:

- Normal replicates

  A normal replicate does not do any kind of consistency checking in view of schema validation across the replication server. It will not verify if the table exists on all replication server or compare the schema. The only verification performed is to check if all participating servers are actually available. The normal replicate will be created as a default if nothing else is specified.

- Mastered replicate

  In contrast to the normal replicate, the mastered replicate defines a master server that checks for all participating servers that the schema, in terms of data types, are equivalent. The schema comparison is done based on the dictionary cache information taken from the other servers. The names of the columns in the participating tables can be different.

- Strict mastered replicates

  The highest level of data consistency in a replicate definition across the replicate servers provides the definition of a strict mastered replicate. In addition to the mastered replicate, the master server also checks the column names.

  There are some limitations when applying schema changes, such as alter tables, on strict mastered replicates, which are otherwise supported by mastered replicates.

To ensure data consistency across the replicate servers, choose *mastered replicate* when defining the replicate. In this case, ER verifies that all the involved tables exist and the attributes of the replicated columns for tables match those on the master replicate server. It also allows you to generate a table automatically on a participant server if the table does not already exist on that server.

Data flow direction

There are two common scenarios of data flow direction that can be defined:

- Primary target

  Primary target replication restricts the data flow in one direction. It defines one replication server as the primary server from where all changes are distributed to the other participating servers. Any changes that are made on the other servers related to the same table and to the defined data set of the replicate are not replicated, but they are allowed.

- Update-anywhere

  Update-anywhere allows the replication of the data changes across the participating server from all sites. This gives the DBA significant flexibility in view of the data flow definition. But it also requires additional planning and definition to manage collisions due to changes on the same row coming from different sources.

Conflict resolution policy

To ensure the understanding of data consistency, the definitions of the possible conflict solutions must be established. There are three types of changes in the ER system that have to be replicated: inserts, deletes, and updates. Typically in a consistent system, a successful insert on the source server also causes a successful insert on the target. But given, for example, the deferred transmission of changes in other replicates or an inconsistent data set on the target, there is still a row with the same primary key. In that case, the incoming insert operation will encounter a collision that has to be handled properly by the database by being defined with the conflict resolution strategy.

ER provides the following conflict resolution strategies:

- Ignore

  With the Ignore conflict resolution strategy defined, the server does not attempt to solve the conflict. In case of a conflict, the row is discarded, and, if defined, a RIS/ATS file is written by the server.

  Table 5-1 summarizes the defined behavior for the Ignore strategy.

  Table 5-1 Apply rules for Ignore conflict resolution

Row exists	Insert	Update	Delete
No	Insert the row.	Discard.	Discard.
Yes	Discard.	Update the row.	Delete the row.

- Always-Apply

  In contrast to the Ignore strategy, Always-Apply always attempts to apply the row in the case of a collision.

  Table 5-2 shows the defined behavior for the Always-Apply strategy.

  Table 5-2 Conflict resolution rules for Always-Apply

Row exists?	Insert	Update	Delete
No	Insert the row.	Insert the row.	N/A.
Yes	Update the row.	Update the row.	Delete the row.

- Time stamp

  The replication server on the target side of a defined replicate solves the collisions based on a comparison of time stamps that are attached to the rows.

To enable the server to do the time stamp comparison, the base table has to be altered or created to apply crcols before the replicate can be created. These are two columns named *cdrserver* and *cdrtime*. The are also referred to as shadow columns.

The cdrtime column contains the time stamp of the last change of the row. In addition, the incoming row also has a time stamp defined so the comparison can be made. Table 5-3 shows the general behavior for the collision handling for time stamp based conflict resolution.

Table 5-3 Conflict resolution rule for time stamp

Row exists	Time stamp	Insert	Update	Delete
No		Insert the row.	Insert the row.	Insert the row into the delete table.
Yes	The time stamp for the changed row is newer than the existing row.	Update the row.	Update the row.	Delete the row.
Yes	The time stamp for the changed row is older than the existing row.	Discard.	Discard.	Discard.
Yes	The time stamps are equal.	Apply row or secondary conflict resolution (SP define invoke this one).	Apply row or secondary conflict resolution (SP define invoke this one).	Apply row or secondary conflict resolution (SP define invoke this one).

An update on a missing row on the target replication server is done in the same way as described for Always-Apply.

There is a special requirement for handling delete operations on all rows in a table participating in a time stamp based replicate. Because a delete removes the row from the target table, there is no possibility to compare the time stamps between an already deleted row and late incoming changes that could be made earlier on the same primary key. To ensure that we do not get ghost rows, replicates with defined time stamp conflict resolution create a shadow table, also called delete table, where the deletes for the rows are kept for time stamp comparison. The schema of the delete table follows the schema as the master table, including all fragmentation definitions.

There is a specific thread named CDRDTCleaner that periodically cleans the expired rows from the shadow tables.

- Stored procedure

  The ER system provides the flexibility to create a SPL-based stored procedure (SP) conflict resolution rule, as a primary conflict resolution rule or as secondary conflict resolution rule to the time stamp conflict resolution rule. When the existing and the new row have the same time stamp, you can establish the conflict resolution policy specified by a stored procedure, to handle this particular case.

  For SP based conflict resolution policy, the replication server requires existing shadow columns on the base table, similar to the time stamp based strategy. Also, with the SP strategy, the replication server automatically creates a shadow table at replicate creation time.

  The SP has to follow some requirements for the input and return values. The following input parameters are expected:

  - Local server name.
  - Local time stamp of the row (NULL when not exists).
  - Does the row exists in the shadow table?
  - Remote server name.
  - Time stamp of the remote operation (the requested change).
  - The operation itself (insert, update, delete).
  - Definition of all columns for the local row.
  - Definition of all columns for the change in the row.

  From the stored procedures, the following values are expected to be returned:

  - The decision of what has to be done with the change in a CHAR data type.
  - A return code for the RIS files to determine the reason why the row was discarded, in an integer data type.
  - The columns that will finally be applied to the target row.

The primary target data flow allows setting the Ignore and the Always-Apply conflict resolution strategy.

For the update-anywhere replicate definition, all conflict resolution strategy rules are allowed. However, be sure that for time stamp and SPL based conflict resolution that the shadow columns are created on the base table.

Columns to be replicated

The replicate definition provides a lot of flexibility with the definition of a projection and a selection list at replicate creation time. The replicate can define which table columns have to be replicated and set filters to restrict the data to be sent.

When to replicate the data

The replicate definition also gives you the flexibility to define when, or in which interval, the data has to be sent out. It makes sense if you have a target server that is only up at scheduled times, or, if the target server is scheduled for some specific activities, such as night processing, where the replicated data is applied, and day processing for the data aggregation of the changes from the previous night.

Floats, triggers, and update columns

There are some miscellaneous configuration options that should be considered when planning a replicate definition. This includes triggers, the handling of floats, and the definition of when, in an update case, all columns or only the changed columns have to be sent to the target.

Floats

ER supports heterogeneous environments in terms of Informix versions and OS types, such as 32-bit, 64-bit, Intel based OSes, or RISC based OSes. In a replication infrastructure where there are different OSs involved, there is a special requirement for handling float values. If a float column has to be replicated across database servers defined on different OSs, the float value has to be converted into an OS independent IEEE format. On the target, the float has be converted back in the internal representation for storage.

Triggers

Triggers defined on the table on the target side of a replicate requires special treatment. With the default definition of a replicate, if the target side has a trigger defined on the table, the trigger will not fire. ER supports the execution of multiple defined triggers for the same event, for example, two update triggers.

In contrast to the general server behavior, where the trigger is only executed (during an update) when the column is changed (on the one on which the trigger is defined), any update on an ER source server, regardless of which column is updated, causes a execution of the trigger.

To achieve the old server behavior, you have to specify in the replicate definition that the source server is not to send the full row at any update. However, you must then consider the side effects for this setting, which are discussed in “Fullrow”.

Fullrow

In general, the replication server sends, in a case of an update of a row, the complete row to the target server. Depending on the selected conflict resolution strategy, this action ensures that the entire row could be inserted when the row does not exist. If there are servers with significant updates in the daily workload, this could have an impact on the network traffic.

The definition of a replicate using `cdr` allows specification with the -f option, which means that only the changes, not all the column values, are sent to the target. This may have a positive impact on the network, which should be monitored by the frequent use of the `onstat -g nif` command.

But there are also some facts to consider. This setting has no effect on the additional ER log space request for updates. Because the logs are generated first, the grouper threads that evaluate the content of the transaction to be sent to the targets run later. Also, the logging of the entire row in the case where the update is a general ER behavior, and specifying that the fullrow send is to be switched off, is a local definition for one replicate.

5.5.2 Defining replicates using OAT

In this section, we provide an example of defining a replicate using OAT.

We plan to define a replicate with following attributes:

- The name of the replicate is *raddr*.
- The replicate is defined on all the columns in the *customer* table in the *stores9* database.
- The server group *newyork* is the master replicate server.
- The conflict resolution strategy is set to *Ignore*.
- The data flow direction is *Update-anywhere*.

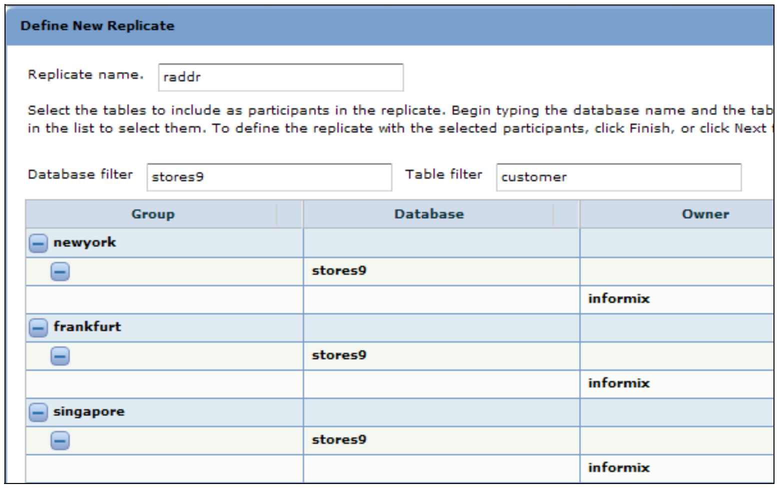
Ensure that at the time of the definition of the replicate, the *customer* table across the involved replication servers is consistent, by verifying that the table exists on all the participants and its column attributes match with that on the master replicate server.

Complete the following steps to define a replicate using OAT:

1. Launch OAT and log into the server. From the menu on the left side, select **Replication** → **Replicates**. On the Actions menu, click **Define Replicate**.

2. In the New Define New Replicate window, define the following items:
   – Input a unique replicate name (in our case, raadr) in the Replicate name text box.
   – You can filter the table list by typing the database name stores9 into the Database filter input field and the table name customer into the Table filter input field.
   – Click **Select** to select the customer table as a participant in the replicate.

   See Figure 5-1 for this part of the Define New Replicate window.



Define New Replicate

Replicate name. raddr

Select the tables to include as participants in the replicate. Begin typing the database name and the tab
in the list to select them. To define the replicate with the selected participants, click Finish, or click Next

Database filter stores9 Table filter customer

Group	Database	Owner
newyork		
	stores9	
		informix
frankfurt		
	stores9	
		informix
singapore		
	stores9	
		informix

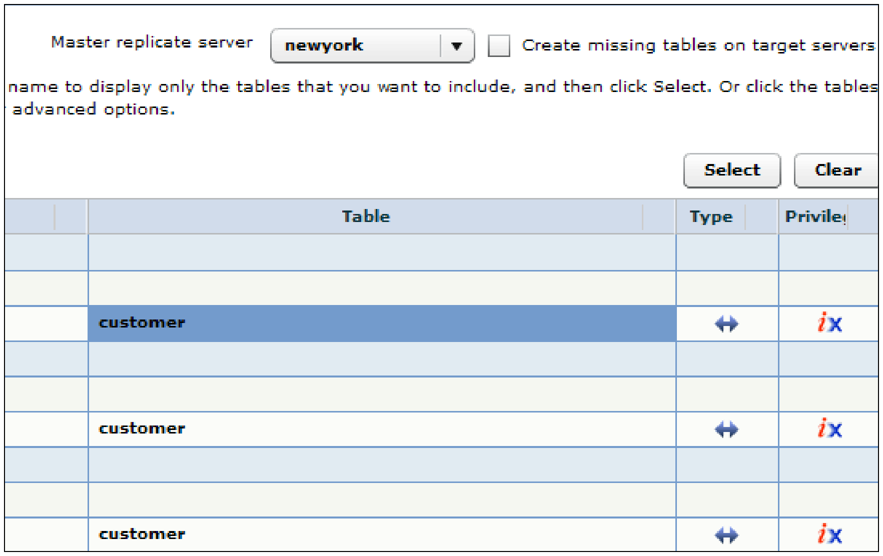
Figure 5-1 Define new replicate

Figure 5-2 on page 122 shows another part of Define New Replicate window:

– Select the **newyork** server group as the basis of the master replicate from the Master replicate server drop-down menu.
– Specify the participant type by clicking the double-arrow icon in the Type column corresponding to the participant table row. The participant type can be:
  • Primary: Each participant both sends and receives data. This is also the default. We specify **Primary** for each participant because we are defining an update-anywhere replicate.
  • Receive-only: The participant only receives data from the primary participants.

  There can be multiple recipients defined in a replicate.

– Specify the privileges used for applying the replicated data. We keep the default privileges. This can be changed by clicking the icon in the Privileges column corresponding to the participant table row.



Master replicate server newyork ▼ ☐ Create missing tables on target servers

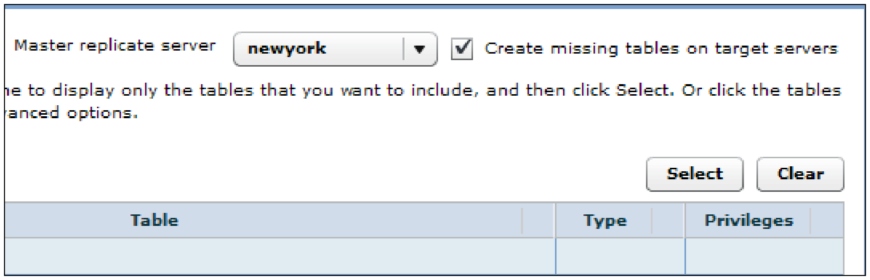
name to display only the tables that you want to include, and then click Select. Or click the tables
advanced options.

Select Clear

	Table	Type	Privile
	customer	↔	ix
	customer	↔	ix
	customer	↔	ix

Figure 5-2 Master replicate server

– You can also specify to have ER create missing tables automatically on target servers by selecting **Create missing tables on target servers**, as shown in Figure 5-3.



Master replicate server newyork ▼ ✓ Create missing tables on target servers

e to display only the tables that you want to include, and then click Select. Or click the tables
anced options.

Select Clear

Table	Type	Privileges

Figure 5-3 Create missing tables on target servers

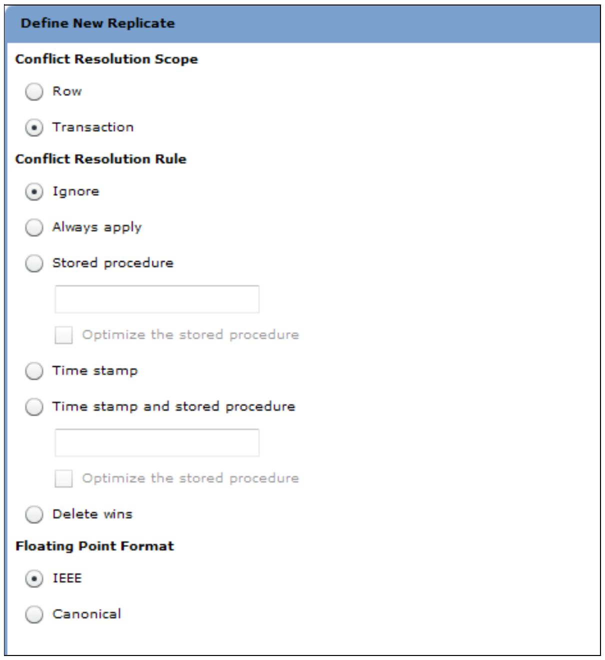
– Click **Next** to proceed to the advanced configuration options window for the replicate.

3. In the advanced configuration options, set the configuration options related to conflict resolution scope, conflict resolution rule, and floating point format.

   – The Conflict Resolution Scope is set to **Transaction** by default.

– The Conflict Resolution Rule is set to **Ignore** by default.
– The Floating Point Format is set to **IEEE** by default.

We accept the default values for these options, as shown in Figure 5-4.


Define New Replicate

Conflict Resolution Scope

- Row
- [✓ Transaction]

Conflict Resolution Rule

- [✓ Ignore]
- Always apply
- Stored procedure
  - Optimize the stored procedure
- Time stamp
- Time stamp and stored procedure
  - Optimize the stored procedure
- Delete wins

Floating Point Format

- [✓ IEEE]
- Canonical

Figure 5-4 Conflict resolution and floating point format options

In the area setting Special Options and Frequency (Figure 5-5), we also use the defaults:

– The Activate Aborted Transaction Spooling (ATS) option is enabled by default.

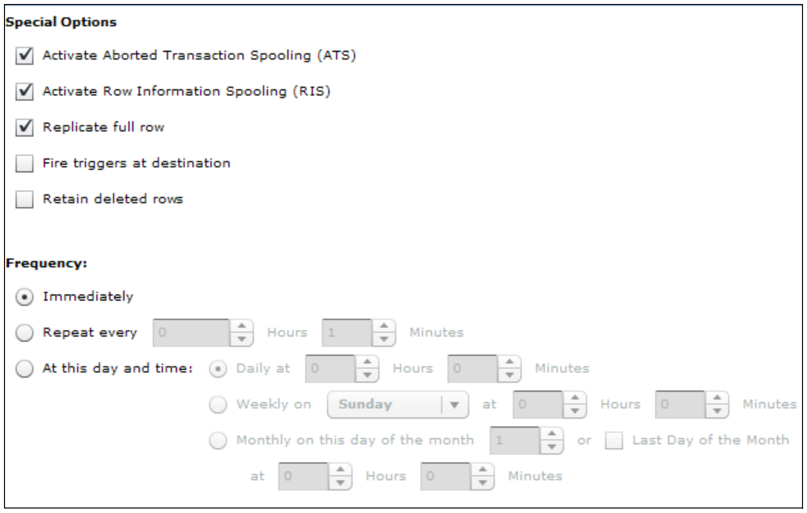
– The Activate Row Information Spooling (RIS) option is enabled by default.

– The Replicate full row option is enabled by default.

– By default, the Fire triggers at destination option specifies that the triggers are not fired at the destination table.

– By default, the Retain deleted rows option specifies that the deleted rows are not retained.

– By default, the Frequency of replication is set to Immediately when data arrives.



Special Options

[✓] Activate Aborted Transaction Spooling (ATS)

[✓] Activate Row Information Spooling (RIS)

[✓] Replicate full row

[ ] Fire triggers at destination

[ ] Retain deleted rows

Frequency:

[●] Immediately

( ) Repeat every 0 Hours 1 Minutes

( ) At this day and time: [●] Daily at 0 Hours 0 Minutes

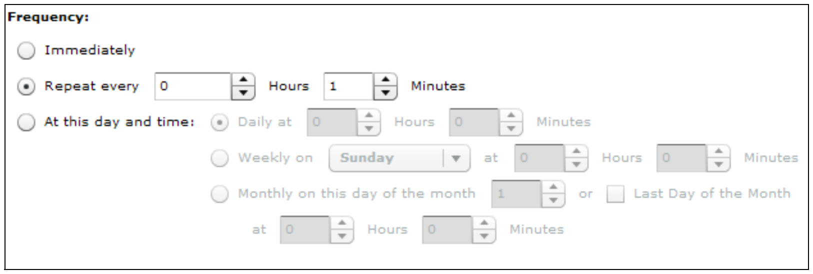
( ) Weekly on Sunday at 0 Hours 0 Minutes

( ) Monthly on this day of the month 1 or [ ] Last Day of the Month

at 0 Hours 0 Minutes

Figure 5-5 Special Options and Frequency

Alternatively, you can specify a frequency for the replicate in hours and minutes by selecting **Repeat every**. Figure 5-6 shows how to specify a frequency of 1 minute.

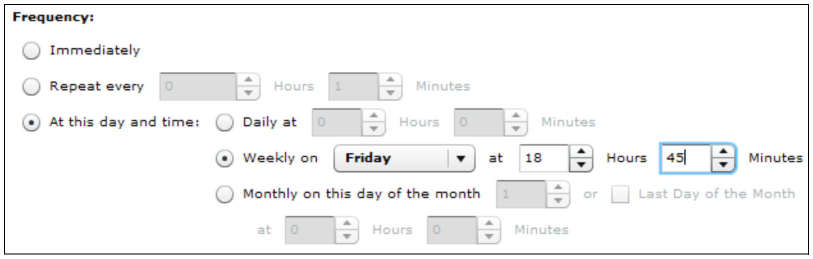


Frequency:

- Immediately
- [✓ Repeat every] 0 Hours 1 Minutes
- At this day and time: [✓ Daily at] 0 Hours 0 Minutes
  - Weekly on Sunday at 0 Hours 0 Minutes
  - Monthly on this day of the month 1 or Last Day of the Month at 0 Hours 0 Minutes

Figure 5-6 Specify a frequency of one minute

Or, you can specify replicating at scheduled time stamp by selecting **At this day and time**. Figure 5-7 shows an example of setting replication Weekly on Friday at 18:45 p.m.

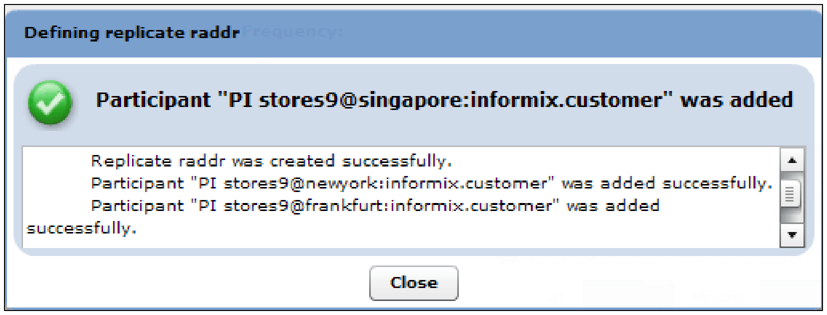


Frequency:

- Immediately
- Repeat every 0 Hours 1 Minutes
- [✓ At this day and time:] Daily at 0 Hours 0 Minutes
  - [✓ Weekly on] Friday at 18 Hours 45 Minutes
  - Monthly on this day of the month 1 or Last Day of the Month at 0 Hours 0 Minutes

Figure 5-7 Specify a time stamp

4. Click **Finish** to create the replicate.
5. A Defining replicate raadr window opens and displays the status of our request. Figure 5-8 shows that the raadr replicate was created successfully.



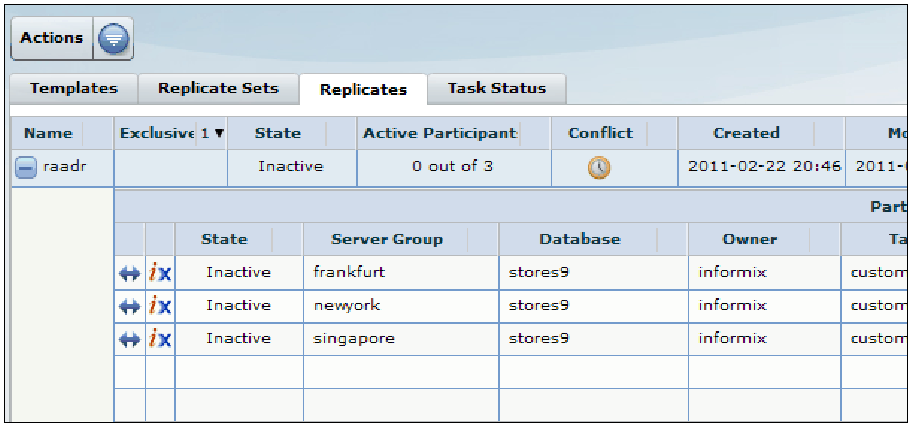
Defining replicate raddr

Participant "PI stores9@singapore:informix.customer" was added

```
        Replicate raddr was created successfully.
        Participant "PI stores9@newyork:informix.customer" was added successfully.
        Participant "PI stores9@frankfurt:informix.customer" was added
successfully.
```

Figure 5-8 Defining a replicate's status

6. Click **Close**, and you are taken back to the Replicates tab, showing the details of the defined replicates. Figure 5-9 shows the newly defined replicate raadr in an Inactive state.



Name	Exclusive 1 ▼	State	Active Participant	Conflict	Created	Mo
raadr		Inactive	0 out of 3		2011-02-22 20:46	2011-

Part

		State	Server Group	Database	Owner	Ta
		Inactive	frankfurt	stores9	informix	custom
		Inactive	newyork	stores9	informix	custom
		Inactive	singapore	stores9	informix	custom

Figure 5-9 Replicate in Inactive state

5.5.3 Defining replicates using cdr

We now describe how to define replicates using the cdr utility. In general, the **`cdr`** command for replicate definition has three major parts:

- Options to define the behavior of the replicate
- The table description for each participating replication server
- The selection and projection list.

Example 5-9 shows a replicate definition for three attached replication servers with several options, to give an impression of how the **`cdr define replicate`** command appears.

Example 5-9 cdr define replicate with various options

```
cdr define replicate -M newyork -C timestamp -S row -I -T -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
```

Note: For the server specification in the selection and projection part of the replicate definition, the server's group name has to be used instead of the server name.

Replicate types

The following examples describe how to define each type of replicate using the cdr replication definition statement.

Normal replicates

Example 5-10 shows the creation of a normal replicate with a verification of the create options.

Example 5-10 cdr define replicate for a normal replicate

```
cdr define replicate -C ignore -S row -A -R raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'

cdr start replicate raadr

#output
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          row,ris,ats,fullrow
REPLID:           323092497 / 0x13420011
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork
```

Mastered replicate

The master replicate is defined by the **`cdr define replicate`** command with the option -M and the specification of a master replication server. An additional option in the **`cdr`** command, "-n n", allows the check of the column names to be switched off.

Example 5-11 shows the definition, and the scope of the definition, of a master replicate. The example shows the **cdr define replicate** command for the creation, and an example of a verification error.

Example 5-11 Behavior of mastered replicates

```
#Table definition on newyork
create table "informix".customer
  (
    customer_code  integer,
    customer_name  char(31),
    company_name   char(20),
    PRIMARY KEY (customer_code)
  ) with crcols;
#table defintion on singapore
create table "informix".customer
  (
    ccustomer_code  integer,
    ccustomer_name  char(31),
    ccompany_name   char(20),
    PRIMARY KEY (ccustomer_code)
  ) with crcols;

cdr define replicate -M newyork -n n -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful

cdr start replicate raadr


#schema with different datatype sizes -- company has a size 25
create table "informix".customer
  (
    ccustomer_code  integer,
    ccustomer_name  char(31),
    ccompany_name   char(25),
    PRIMARY KEY (ccustomer_code)
  ) with crcols;

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started

Master Replicate Error
    Column mismatch

Master:         company_name    char(20)
Participant:    ccompany_name   char(25)

1 Error(s)
Verification of stores9@singapore:informix.customer failed
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
Master Replicate Verification Total errors:1
```

Again, the defined replicates can be verified with the **`cdr list replicate`** command. Example 5-12 shows the output of that verification. Pay particular attention to the last row.

Example 5-12 Verification for a mastered replicate

```
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Timestamp
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          transaction,fullrow
REPLID:           323092500 / 0x13420014
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork
REPLTYPE:         Master
```

Strict mastered replicates

The definition of a strict master replicate is specified with the "-n y" option. Similar to the mastered replicate, the master server has to specified after the -M option.

In Example 5-13, we use the same schema difference introduced in Example 5-11 on page 128. In contrast to the behavior for the mastered replicate definition, where we only report an error for the differences in the data types, we will see additional errors when different column names are used in the participating tables for a strict mastered replicate definition.

Example 5-13 Create a mastered replicate

```
cdr define replicate -M newyork -n y -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
```

```
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started

Master Replicate Error
    Column name mismatch

Master:         customer_code
Participant:    ccustomer_code

Master Replicate Error
    Column name mismatch

Master:         customer_name
Participant:    ccustomer_name

Master Replicate Error
    Column mismatch

Master:         company_name    char(20)
Participant:    ccompany_name   char(25)

3 Error(s)
Verification of stores9@singapore:informix.customer failed
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
Master Replicate Verification Total errors:1
```

Unfortunately, when trying to verify if the replicate is a strict mastered replicate, the **`cdr list replicate`** output is not different from the output of other replicates. For verification here, use the **`onstat -g cat`** command. Look at the line starting with “Column Name Verification” to see the difference for both types of replicates. For strict mastered replicates, this is set to “ON”.

Data flow direction

The following examples describe how to specify the data flow direction using the cdr replication definition statement.

Primary target

The role of the server that is either to act as a primary or as a recipient for the data has to be identified by the letter P or R in the table specification. Example 5-14 shows the definition of a primary target replicate.

Example 5-14 Define a primary target replicate with two servers

```
#define primary target replicate
cdr define replicate -M london -C ignore raadr "P
stores9@london:informix.customer" "select * from informix.customer"  "R
stores9@frankfurt:informix.customer" "select * from informix.customer"

#output
Verification of stores9@london:informix.customer started
Verification of stores9@london:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful

cdr start replicate raadr

#update on the primary on london
dbaccess -e stores9 << EOF
update customer set customer_name="James Henry" where customer_code=128;
EOF
#on frankfurt
dbaccess -e stores9 << EOF
select customer_name from customer where customer_code=128;
EOF

#output
customer_name  James Henry

#update on a recipient on frankfurt
dbaccess -e stores9 << EOF
update customer set customer_name="Jason Wallack" where customer_code=127;
EOF
#on london
dbaccess -e stores9 << EOF
select customer_name from customer where customer_code=127;
EOF

#output - nothing has changed
customer_name  Kim Satifer
```

Update-anywhere

The update-anywhere scenario is the default behavior in the **`cdr create replicate`** command. Example 5-15 shows the details of how to specify an update-anywhere scenario. Any changes on the newyork server are replicated to the other servers. This is also true for all changes that are made on the frankfurt and singapore servers.

Example 5-15 Define an update-anywhere replicate

```
cdr define replicate -C timestamp raadr  "stores9@newyork:informix.customer"
"select * from informix.customer"  "stores9@singapore:informix.customer"
"select * from informix.customer" "stores9@frankfurt:informix.customer" "select
* from informix.customer"
```

Conflict resolution policy

The conflict resolution in the **`cdr define replicate`** command has to specified with the -C option.

To show the differences in behavior for the different strategies, the same collision scenario is used. And, to keep the example simple, the update of a not-existing row on the target side is used.

Ignore

Example 5-16 shows the definition of a replicate with a conflict resolution set to Ignore. The scope for the replicate set is defined as a transaction. After the definition of the replicate, there is an attempt to update a not-existing row that generates an ATS on the target.

Example 5-16 Create a replicate with a Ignore conflict resolution

```
cdr define replicate -C ignore -S trans -A raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

cdr list replicate raadr

#output
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
```

```
PARTICIPANT:       stores9:informix.customer
OPTIONS:           transaction,ats,fullrow
REPLID:            323092488 / 0x13420008
REPLMODE:          PRIMARY  ON:newyork
APPLY-AS:          INFORMIX ON:newyork

dbaccess -e stores9 <<
update customer set customer_name="Frank Lessor" where customer_code=101;
EOF

# ATS file
TXH Source ID:4930 / Name:newyork / CommitTime:11-02-21 19:09:04
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-02-21 19:09:05
TXH Number of rows processed when transaction was aborted:1
TXH One or more rows in a transaction defined with tx scope were rejected
TXH CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 /
ISAM:0
----------
RRH Row:1 / Replicate Id: 323092488 / Table: stores9@informix.customer /
DbOp:Update
RRD 101|Frank Lessor|All Sports Supplies
```

Always-Apply

With Always-Apply, when there is an attempt to update a not-existing row on the target, the server converts the update into an insert. That behavior is shown in Example 5-17.

Example 5-17 Define the replicate with Always-Apply and a collision situation

```
cdr define replicate -C always -S trans -A raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr list replicate raadr

#output
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:         raadr
STATE:             Active ON:newyork
CONFLICT:          Always Apply
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       stores9:informix.customer
OPTIONS:           transaction,ats,fullrow
REPLID:            323092487 / 0x13420007
REPLMODE:          PRIMARY  ON:newyork
```

```
APPLY-AS:         INFORMIX ON:newyork

#On new york update a row which doesnt exists on singapore
dbaccess -e stores9 << EOF
update customer set customer_name="Frank Lessor" where customer_code=101;
EOF

#On singapore
dbaccess -e stores9 << EOF
select * from customer where customer_code=101;
EOF

#output
customer_code  101
customer_name  Frank Lessor
company_name   All Sports Supplies
```

Be aware that the defined behavior for collisions of Update-Apply is only valid for replicates that send the full set of columns to the target with the -f y option. For replicates sending only the updated rows, the server generates a RIS file.

Time stamp

Two concurrent updates are used in Example 5-18 to demonstrate a common collision scenario. The example has defined a replicate with a replication time of Wednesday at 16:20. Given there are multiple servers that have updated the same row, with different time stamps, the time stamp based collision ensures that the latest version of the row is the final version on the target.

Example 5-18 Time stamp based collision scenario

```
cdr define replicate -C timestamp -a Wed.16:20 -S trans -R -A raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'

cdr start replicate raadr

#on newyork at 16:11
dbaccess -e stores9 << EOF
update customer set customer_name="16:11" where customer_code=101;
EOF

#on singapore at 16:14
dbaccess -e stores9 << EOF
update customer set customer_name="16:14" where customer_code=101;
EOF

#on frankfurt at 16:20
```

```
dbaccess -e stores9 << EOF
select * from customer where customer_code=101;
EOF

#output
customer_code customer_name                    company_name

          101 16:14                            All Sports Supplies
```

Stored procedure

Example 5-19 shows how to actually create a replicate with a stored procedure conflict resolution. In addition, a sample stored procedure is presented to show which parameters have to be specified.

The stored procedure in the example demonstrates the implementation of the time stamp based conflict resolution. The particular example can be changed easily to adapt to special business requirements, such as do not allow deletes on the target, or only allow changes in certain primary key ranges in general or from the certain source server, or only allow changes based on certain times from the particular source server. Otherwise, a mixed strategy can be defined for applying changes to existing and not-exiting rows or to combine ignore and time stamp based resolution.

Example 5-19 Stored procedure based replicates

```
#create the stored procedure on all replication server
dbaccess << EOF
database stores9;
create procedure conflict_acct_sp
    (
--required server parameter for the change and the existing row attrib
    localServer CHAR(18),
    localTS DATETIME YEAR TO SECOND,
    localDelete CHAR(1),
    replServer  CHAR(18),
    replTS  DATETIME YEAR TO SECOND,
    replDbop CHAR(1),

--define all the local row columns
    local_customer_code LIKE customer.customer_code,
    local_customer_name LIKE customer.customer_name,
    local_company_name LIKE customer.company_name,

--define all the remote changes on the row
     remote_customer_code LIKE customer.customer_code,
     remote_customer_name LIKE customer.customer_name,
     remote_company_name LIKE customer.company_name
)

RETURNING
--return an Operation what to do with the change , and a reason code
     CHAR(1), INT,
--return the values for the to be applied changes here for customer tab
    INTEGER, CHAR(31), CHAR(20);

{-------------------------------------------------------------------------}
-- row does exists decision depending on timestamp comparison

 IF localTS is NOT NULL THEN
    IF ((replTS > localTS) OR
        ((replTS = localTS) AND
         (replServer < localServer)))
        THEN
-- apply the change
        RETURN 'A', 600,
            remote_customer_code, remote_customer_name, remote_company_name;

    ELSE
-- reject the change request RIS file will be created
        RETURN 'O', 700,
            local_customer_code, local_customer_name, local_company_name;

    END IF
-- local row doesnt exists -- Apply the row
 ELIF localTS IS NULL THEN
        RETURN 'A', 800,
            remote_customer_code, remote_customer_name, remote_company_name;

 END IF

END PROCEDURE;

#create a replicate with a SP conflict resolution, SP is named conflict_acct_sp
cdr define replicate -C conflict_acct_sp -S trans -A spl
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate spl

#Verification
cdr list replicate spl

#output
DEFINED REPLICATES ATTRIBUTES
------------------------------
```

```
REPLICATE:          spl
STATE:              Active ON:newyork
CONFLICT:           conflict_acct_sp
FREQUENCY:          immediate
QUEUE SIZE:         0
PARTICIPANT:        stores9:informix.customer
OPTIONS:            transaction,ats,fullrow
REPLID:             323092496 / 0x13420010
REPLMODE:           PRIMARY  ON:newyork
APPLY-AS:           INFORMIX ON:newyork

#make an inconsistency on singapore -- delete some rows
dbaccess -e stores9 << EOF
begin work without replication;
delete from customer where customer_code<110;
commit work;
EOF

#Update on frankfurt non existing row on the source
dbaccess -e stores9 << EOF
update customer set customer_name=”AAAAA” where customer_code=101;
EOF

#On singapore
#SP detects row doesnt exists because locals is null,
#change will be applied -- means the update is converted in a Insert
dbaccess -e stores9 << EOF
select from customer where customer_code=101;
EOF

#output
customer_code customer_name                    company_name

          101 AAAAA                            All Sports Supplies
```

Which columns will be replicated

The specification of which columns will be replicated can be done in the selection part of the **cdr define replicate** command.

In Example 5-20, only the columns customer_code and customer_name are part of the replicate. Any changes applied to those columns are sent out, and changes to any other columns are applied locally but are not replicated.

Example 5-20 Define a projection in the replicate

```
cdr define replicate -T -C timestamp -S row -A -R raadr
"stores9@newyork:informix.customer" "select customer_code,customer_name from
informix.customer" "stores9@singapore:informix.customer" "select
```

```
customer_code,customer_name from informix.customer"
"stores9@frankfurt:informix.customer" "select customer_code,customer_name from
informix.customer"

cdr start replicate raadr

#on newyork
dbaccess -e stores9 << EOF
select * from customer where customer_code=101;

#output
customer_code customer_name                  company_name

          101 Ludwig Pauli                   All Sports Supplies

update customer set customer_name="Jason Wallack" where customer_code=101;
update customer set company_name="City Sports" where customer_code=101;
EOF

#output on frankfurt
customer_code customer_name                  company_name

          101 Jason Wallack                  All Sports Supplies
```

The replication behavior in the case of the specification of a selection list is similar. Here the DBA is able to apply a filter in the WHERE clause of the select definition. Only rows that meet the filter conditions are replicated. Projection and selection can also be mixed in the replication definition, as shown in Example 5-21.

Example 5-21 Projection and selection in replicates

```
cdr define replicate -C timestamp raadr  "stores9@newyork:informix.customer"
"select customer_code,customer_name from informix.customer where
customer_code=101" "stores9@singapore:informix.customer" "select
customer_code,customer_name from informix.customer where customer_code=101"
"stores9@frankfurt:informix.customer" "select customer_code,customer_name from
informix.customer where customer_code=101"

cdr start replicate raadr

#on newyork run the following updates
dbaccess -e stores9 << EOF
select * from customer where customer_code=101;
select * from customer where customer_code=102;
EOF

#output
```

```
customer_code customer_name                  company_name

          101 Ludwig Pauli                   All Sports Supplies

customer_code customer_name                  company_name

          102 Carole Sadler                  Sports Spot

dbaccess -e stores9 << EOF
update customer set customer_name="Jason Wallack" where customer_code=102;
update customer set customer_name="Marvin Hanlon",company_name="Bay Sports"
where customer_code=101;
EOF

#output on frankfurt
customer_code customer_name                  company_name

          101 Marvin Hanlon                  All Sports Supplies

customer_code customer_name                  company_name

          102 Carole Sadler                  Sports Spot
```

When to replicate the data

Example 5-22 shows how to specify an interval for the replicate with the -e option and is measured in minutes. The default for the send interval is defined as 0. This means that the prepared transactions in the send queue is immediately sent out to the target without waiting.

Example 5-22 Specify an interval for replication data

```
cdr define replicate -e 1  -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr list replicate raadr

#output
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Timestamp
FREQUENCY:        every 00:01
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          transaction,fullrow
```

```
REPLID:          323092495 / 0x1342000f
REPLMODE:        PRIMARY  ON:newyork
APPLY-AS:        INFORMIX ON:newyork
```

Another option to schedule replication time, rather than set a specific interval for the replication, is setting a certain time stamp. This mean all transactions that need to be sent to particular targets specified by the replicate with the time stamp schedule are kept on hold until the time stamp is reached. This task can be done by using the additional option -a in the replicate definition.

Setting this option will request, depending on the volume of work on the replicates base tables, additional disk space. The dbspaces defined with the CDR_QHDR_DBSPACE and CDR_QDATA_SBSPACE **onconfig** parameters store the transactions and maintenance information. This is because most likely the transactions in these types of replicates need to be spooled to disk to prevent transaction blocking. Example 5-23 defines a replicate with a scheduled replication time and verifies the replicate with **cdr list replicate**.

Example 5-23 Define a replicate replicating data at a scheduled time stamp

```
cdr define replicate -M newyork -a fri.18:45 -C timestamp raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

DEFINED REPLICATES ATTRIBUTES
------------------------------
REPLICATE:       raadr
STATE:           Active ON:newyork
CONFLICT:        Timestamp
FREQUENCY:       week Fri.18:45
QUEUE SIZE:      0
PARTICIPANT:     stores9:informix.customer
OPTIONS:         transaction,fullrow
REPLID:          323092501 / 0x13420015
REPLMODE:        PRIMARY  ON:newyork
APPLY-AS:        INFORMIX ON:newyork
REPLTYPE:        Master
```

Creating missing tables automatically

There is a -u option in the **`cdr define replicate`** command where a DBA can specify that on all replication servers where a base table for a replicate does not exist, the table is created automatically at replicate creation time. This provides a comfortable way to spread a defined table schema across the database server in a consistent way. Example 5-24 demonstrates the automatic creation of the table that does not exist on singapore in combination with a mastered replicate.

Example 5-24 Create a table automatically at replicate definition time

```
cdr define replicate -M newyork -u -C timestamp -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"
"stores9@london:informix.customer" "select * from informix.customer"

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Creating table...
create table informix.customer (
        customer_code   integer not null,
        customer_name   lvarchar(31),
        company_name    char(20),
        primary key (customer_code)) with CRCOLS;

#output
Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
Verification of stores9@london:informix.customer started
Verification of stores9@london:informix.customer is successful
```

This process also works for a replicate definition with a projection list. In this definition, only the replicated columns taken from the projection list will be created in the table, as shown in Example 5-25. Also, you might notice that in case of a time stamp based replication, the `with CRCOLS` clause in the create table statement is automatically added.

Example 5-25 Create a table based on a replicates projection list

```
cdr define replicate -M newyork -T -u -C timestamp -S row -A -R raadr
stores9@newyork:informix.customer 'select customer_code,company_name from
informix.customer where customer_code=101' stores9@singapore:informix.customer
'select customer_code,company_name from informix.customer where
customer_code=101' stores9@frankfurt:informix.customer 'select
customer_code,company_name from informix.customer where customer_code=101'
```

```
stores9@london:informix.customer 'select customer_code,company_name from
informix.customer where customer_code=101'

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Creating table...
create table informix.customer (
        customer_code   integer not null,
        company_name    char(20),
        primary key (customer_code)) with CRCOLS;

Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
Verification of stores9@london:informix.customer started
Verification of stores9@london:informix.customer is successful
```

Floats, triggers, and update columns

Here is a brief discussion of setting these miscellaneous configuration options during replicate definition using the cdr utility.

Floats

The -I option during replicate definition can be specified to transfer replicated floating-point numbers in IEEE format. Without this definition, the representation of the float on the target system can be invalid and the original meaning of the value destroyed.

Triggers

The execution of triggers on the target have to be specified at definition time of the replicate with the -T option.

Example 5-26 shows how to enable a trigger on a target table.

Example 5-26 Enable a trigger on a target table in a replication definition

```
#on singapore a trigger is defined
dbaccess -e stores9 << EOF
drop table "informix".customer_log;

create table "informix".customer_log
  (
    customer_code integer,
    customer_name char(31),
    company_name char(20),
    update_time datetime year to minute,
    old_customer_name char(31)
  );

create trigger "informix".updt update of customer_name on
"informix".customer referencing old as pre_upd new as post_upd
for each row
(
    insert into "informix".customer_log
       (customer_code,customer_name,company_name,update_time,old_customer_name)
values
       (pre_upd.customer_code,pre_upd.customer_name,pre_upd.company_name ,
        CURRENT year to fraction(3),post_upd.customer_name)
);

EOF

#define the replicate
cdr define replicate -T -C timestamp raadr "stores9@newyork:informix.customer"
"select * from informix.customer" "stores9@singapore:informix.customer" "select
* from informix.customer" "stores9@frankfurt:informix.customer" "select * from
informix.customer" "stores9@london:informix.customer" "select * from
informix.customer"

cdr start replicate raadr

#on newyork
dbccess -e stores9 << EOF
update customer set customer_name="Jason Wallack" where customer_code=101;
update customer set company_name="City Sports" where customer_code=101;
EOF

#on singapore
dbaccess << EOF
select count(*) from customer_log;
EOF

#output
(count(*))

2
```

Fullrow

Take the case where the Always-Apply conflict resolution strategy is defined, and an update on a target arrives and the row does not exist. In the default definition of this strategy, the row is inserted instead of updated. In case you have set both Always-Apply and fullrow to off, the update on the target will be discarded. Example 5-27 shows the differences in the behavior.

Example 5-27 Sending the full row versus only the changed columns with Always-Apply

```
#On singapore - create an inconsistency between the servers
dbaccess -e stores9 << EOF
begin work without replication;
delete from customer where customer_code=128;
commit work;
EOF

#define a replicate with full row on
cdr define replicate -M newyork -f y -C always -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#on newyork do the update
dbaccess -e stores9 << EOF
update customer set customer_name="Jason Wallack" where customer_code=128;
EOF

#on singapore verify the change
dbaccess -e stores9 << EOF
select * from customer where customer_code=128;
EOF

#output
customer_code customer_name                   company_name

          128 Jason Wallack                   Phoenix University

#on singapore a RIS file will be written
TXH Source ID:4930 / Name:newyork / CommitTime:11-03-02 17:50:53
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-03-02 17:50:53
----------
RRH Row:1 / Replicate Id: 323092494 / Table: stores9@informix.customer / DbOp:Update
RRH CDR:3 (Warning: Update exception, row does not exist in target table, converted to
insert) / SQL:0 / ISAM:0
RRD ||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

```
-------------------------------------------------------------------------------

#On singapore - create an inconsistency between the servers
dbaccess -e stores9 << EOF
begin work without replication;
delete from customer where customer_code=128;
commit work;
EOF

#create a replicate with full row off
cdr define replicate -M newyork -f n -C always -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#On newyork do the update
dbaccess -e stores9 << EOF
update customer set customer_name="Jason Wallack" where customer_code=128;
EOF

#on singapore
dbaccess -e stores9 << EOF
select * from customer where customer_code=128;
EOF

#output
customer_code customer_name                    company_name

No rows found.

#online log on singapore
17:58:32  CDR CDRD_13: transaction aborted (All rows in a transaction defined with row
scope were rejected) with sql error 0 isam error 0.
17:58:32  CDR CDRD_13: failed rows spooled to file
/files/IDS/RIS/ris.singapore.newyork.D_13.110302_17:58:32.1
17:58:32  CDR CDRD_13: failed transaction spooled to file
/files/IDS/ATS/ats.singapore.newyork.D_13.110302_17:58:32.2

#RIS file will be written on singapore
TXH Source ID:4930 / Name:newyork / CommitTime:11-03-02 17:58:31
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-03-02 17:58:32
----------
RRH Row:1 / Replicate Id: 323092495 / Table: stores9@informix.customer / DbOp:Update
RRH CDR:3 (Warning: Update exception, row does not exist in target table, converted to
insert) / SQL:0 / ISAM:0
RRD 128|Jason Wallack|?
==========
TXH Transaction aborted
TXH ATS file:/files/IDS/ATS/ats.singapore.newyork.D_13.110302_17:58:32.2 has also been
created for this transaction

#ATS file will be written on singapore
TXH RIS file:/files/IDS/RIS/ris.singapore.newyork.D_13.110302_17:58:32.1 has also been
created for this transaction
==========
TXH Source ID:4930 / Name:newyork / CommitTime:11-03-02 17:58:31
TXH Target ID:4936 / Name:singapore / ReceiveTime:11-03-02 17:58:32
TXH Number of rows processed when transaction was aborted:1
TXH All rows in a transaction defined with row scope were rejected
TXH CDR:3 (Warning: Update exception, row does not exist in target table, converted to
insert) / SQL:0 / ISAM:0
----------
RRH Row:1 / Replicate Id: 323092495 / Table: stores9@informix.customer / DbOp:Update
RRD 128|Jason Wallack|?
```

5.5.4 Some common mistakes

In this section, we describe a few possible configuration errors that might be encountered when defining replicates using the cdr utility.

Master server definition forgotten

The first example is the attempt to define a master replicate with a -M option, but without specification of the master server. This is shown in Example 5-28.

Example 5-28 Incorrect master server definition

```
cdr define replicate -M -T -u -C timestamp -S row -A -R raadr
stores9@newyork:informix.customer 'select customer_num,fname from
informix.customer where customer_num=1' stores9@singapore:informix.customer
'select customer_num,fname from informix.customer where customer_num=1'
stores9@frankfurt:informix.customer 'select customer_num,fname from
informix.customer where customer_num=1'

#output
Master node '-T' was not specified in the list of replicate participants
```

Replicates with databases missing on the replicate server

An attempt to create a normal replicate on a server where the database does not exist, generates an error. Later in the use of the replicate, the error appears in the log file, as shown in Example 5-29.

Example 5-29 Missing database on the targets

```
cdr define replicate -C timestamp -S row -A -R -f n raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
```

```
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

$onstat -m
20:05:37  CDR GC peer request failed: command: peer process failed, error 6, CDR server
4930
20:05:55  CDR GC peer request failed: command: peer process failed, error 6, CDR server
4930
20:06:28  CDR GC peer request failed: command: peer process failed, error 30, CDR server
4930
20:07:23  Can Not Create TAB2List in ReplRefConCtor
20:07:23  ReplRefCon_tdCtor failed sqerr: -329 iserr: -111.

20:07:23  Can not get replmap for replicate 323092516
20:07:23  CDR CDRDS: transaction aborted (Error in retrieving the replicate's
attributes) with sql error 0 isam error 0.
```

Quite different, but more obvious, is the error in attempting the same definition, but the database does not exist on the connection server. This is shown in Example 5-30.

Example 5-30   Database does not exist on the connection server

```
cdr define replicate...

#output
failed -- database does not exist  (6)
```

An attempt to create a mastered replicate, where the table or the database does not exist on one of the targets, generates the output shown in Example 5-31. The use of the -u option to create the missing table would solve the problem.

Example 5-31   Masted replicate and missing objects

```
#missing database
cdr define replicate -M newyork -u -C timestamp -S row -A -R raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'
stores9@london:informix.customer 'select * from informix.customer'

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
connect to stores9@singapore failed
Could not do a physical-order read to fetch next row.
 (-244)
command failed -- unable to connect to server specified  (5)
```

```
#missing table
cdr define replicate -M newyork -C timestamp -S row -A -R raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'
stores9@london:informix.customer 'select * from informix.customer'

Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
table does not exist  - stores9@singapore.informix:customer
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
Verification of stores9@london:informix.customer started
Verification of stores9@london:informix.customer is successful
Master Replicate Verification Total errors:1
```

Schema mismatch

The definition of a normal replicate does not check the schema of the tables on all the replication servers. In case one of the tables has a quite different schema, for example, one of the char fields is quite longer than on all other tables, a short time after the replicate is activated, the server on the target side starts reporting RIS files (if enabled).

Primary target and time stamp conflict resolution

A primary target data flow definition only allows the settings ignore or Apply-Always conflict resolution option. An attempt to set a time stamp base conflict resolution causes the error shown in Example 5-32.

Example 5-32 Time stamp and primary target are not allowed

```
cdr define replicate -M newyork -C timestamp -S row -A -R raadr 'P
stores9@newyork:informix.customer' 'select * from informix.customer' 'R
stores9@singapore:informix.customer' 'select * from informix.customer' 'R
stores9@frankfurt:informix.customer' 'select * from informix.customer'

#output
Verification of stores9@newyork:informix.customer started
Verification of stores9@newyork:informix.customer is successful
Verification of stores9@singapore:informix.customer started
Verification of stores9@singapore:informix.customer is successful
Verification of stores9@frankfurt:informix.customer started
Verification of stores9@frankfurt:informix.customer is successful
command failed -- conflict mode for replicate not ignore or always apply (69)
participant stores9@singapore:informix.customer 'select  customer_code , customer_name ,
company_name from informix.customer'
```

Using the server name instead of the group name

For the definition of the table in the replicate, the table has to be specified with the group name of the server, the owner, and the table name. If instead of the group name the DBSERVERNAME is used, the **`cdr define replicate`** command fails with the error shown in Example 5-33. The group name for the first table was newyork, but the DBSERVERNAME er_prim_11 was specified. You can also refer to the general `sqlhosts` settings in the example environment shown in Example 5-33.

Example 5-33 Using server name instead of group name

```
cdr define replicate -C timestamp -S row -A -R raadr
stores9@er_prim_11:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'

#output
command failed -- undefined server  (37)
participant stores9@er_prim_11:informix.customer 'select * from
informix.customer'
```

Invalid operation on a leaf server

The creation of replicates is only allowed on root and non-root replication servers. Any attempt to create a replicate on a leaf replication server causes an error and the replicate is not created. A typical error message is shown in Example 5-34.

Example 5-34 Unsuccessful attempt to create a replicate on a leaf server

```
#On singapore
cdr define replicate -C timestamp -S row raadr
stores9@newyork:informix.customer 'select * from informix.customer'
stores9@singapore:informix.customer 'select * from informix.customer'
stores9@frankfurt:informix.customer 'select * from informix.customer'

#output
command failed -- request denied on limited server
(75)
```

5.5.5 Error handling, scope, and ATS and RIS files

For the definition of the error handling behavior in the replicate, the scope of the error handling and the specification of writing aborted transaction spooling (ATS) and row information spooling (RIS) files belong together. In general, transactions are applied to the target server and the expectation is that the apply will be successful.

Depending on the defined conflict resolution strategy for the replicate in case of differences in the table data between the source and the target, the application of the changes to the data by the database server can fail. ER defines, at this point for the target database server, two different options that are specified by the scope in the replicate.

The scope can define that only the failed row operation is skipped and all the other possible successful operations in the current transaction are applied. There are risks in the final data consistency of this setting. Otherwise, to keep transactional consistency on the target server, the error handling scope can be defined in view of the transaction. At the occurrence of the first error during the application of the actual transaction, the server keeps the transaction from applying to the target and rolls back all of the error time stamps of already applied changes.

The scope will be specified with the -S option. The patterns, row or trans, defines which kind of scope is used.

Any error in the apply of transactions or rows can be reported in spool files called ATS or RIS files. In addition to the **`cdr define server`** command, where the location of the spool files is defined, the replicate can switch writing the spool files on or off. Enabling the spool file writing allows the DBA to manually check the circumstances for the errors and to repair them.

The ATS spooling is switched on with the -A option, and the RIS spooling with the -R option.

To illustrate the different results depending on the scope, Example 5-35 on page 151 and Example 5-36 on page 152 show the server behavior with a defined replicate that has conflict resolution set to ignore. In both cases, the customer table is used. There are 28 rows in the table with a range of 101-128 for the customer_code. To make the table inconsistent across the replication servers, we delete customer_code 128 on the newyork server without replicating the change. After that, we inserted five additional rows into the table within a single transaction.

Example 5-35 shows the row scope, where the server has added four new rows and only the failed row is rejected.

Example 5-35   Row scope and the final results in an error case

```
#define row scope
cdr define replicate -C ignore -S row -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#verification
cdr list replicate raadr

#output
DEFINED REPLICATES ATTRIBUTES
------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          row,ris,ats,fullrow
REPLID:           323092487 / 0x13420007
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork

#on newyork
dbaccess -e stores9 << EOF
begin work without replication;
delete from customer where customer_code=128;
commit  work;

begin work ;
insert into customer( customer_code,company_name) values ( 128,"AAAAA");
insert into customer( customer_code,company_name) values ( 129,"BBBBB");
insert into customer( customer_code,company_name) values ( 130,"BBBBB");
insert into customer( customer_code,company_name) values ( 131,"CCCCC");
insert into customer( customer_code,company_name) values ( 132,"CCCCC");
commit work;
EOF

#on frankfurt
dbaccess -e stores9 <<
select * from customer where customer_code>127;
EOF
```

```
#output
customer_code customer_name                   company_name

          128 Frank Lessor                    Phoenix University
          129                                 BBBBB
          130                                 BBBBB
          131                                 CCCCC
          132                                 CCCCC

# RIS file
TXH Source ID:4930 / Name:newyork / CommitTime:11-03-02 16:44:10
TXH Target ID:4932 / Name:frankfurt / ReceiveTime:11-03-02 16:44:11
----------
RRH Row:1 / Replicate Id: 323092487 / Table: stores9@informix.customer / DbOp:Insert
RRH CDR:5 (Error: Insert aborted, row already exists in target table) / SQL:0 / ISAM:0
LRD 128|Frank Lessor|Phoenix University
RRD 128||AAAAA
==========
TXH Transaction committed
TXH Total number of rows in transaction:5
```

Example 5-36 illustrates the transaction scope where the server rejected the insert for all of the rows.

Example 5-36 Transaction scope and the results in an error case

```
#define row scope
cdr define replicate -C ignore -S trans -A -R raadr
"stores9@newyork:informix.customer" "select * from informix.customer"
"stores9@singapore:informix.customer" "select * from informix.customer"
"stores9@frankfurt:informix.customer" "select * from informix.customer"

cdr start replicate raadr

#verification
cdr list replicate raadr

#output
DEFINED REPLICATES ATTRIBUTES
------------------------------
REPLICATE:        raadr
STATE:            Active ON:newyork
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          transaction,ris,ats,fullrow
REPLID:           323092490 / 0x1342000a
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork
```

```
#on newyork
dbaccess -e stores9 <<
begin work without replication;
delete from customer where customer_code=128;
commit  work;

begin work ;
insert into customer( customer_code,company_name) values ( 128,"AAAAA");
insert into customer( customer_code,company_name) values ( 129,"BBBBB");
insert into customer( customer_code,company_name) values ( 130,"BBBBB");
insert into customer( customer_code,company_name) values ( 131,"CCCCC");
insert into customer( customer_code,company_name) values ( 132,"CCCCC");
commit work;

#On frankfurt
dbaccess -e stores9 <<
> select * from customer where customer_code>127;
EOF

customer_code customer_name                  company_name

          128 Frank Lessor                   Phoenix University

#online.log
17:01:57  CDR CDRD_8: transaction aborted (One or more rows in a transaction defined
with tx scope were rejected) with sql error 0 isam error 0.
17:01:57  CDR CDRD_8: failed rows spooled to file
/files/IDS/RIS/ris.frankfurt.newyork.D_8.110302_17:01:56.1
17:01:57  CDR CDRD_8: failed transaction spooled to file
/files/IDS/ATS/ats.frankfurt.newyork.D_8.110302_17:01:56.2



#ATS file
TXH RIS file:/files/IDS/RIS/ris.frankfurt.newyork.D_8.110302_17:01:56.1 has also been
created for this transaction
==========
TXH Source ID:4930 / Name:newyork / CommitTime:11-03-02 17:01:56
TXH Target ID:4932 / Name:frankfurt / ReceiveTime:11-03-02 17:01:56
TXH Number of rows processed when transaction was aborted:1
TXH One or more rows in a transaction defined with tx scope were rejected
TXH CDR:5 (Error: Insert aborted, row already exists in target table) / SQL:0 / ISAM:0
----------
RRH Row:1 / Replicate Id: 323092490 / Table: stores9@informix.customer / DbOp:Insert
RRD 128||AAAAA
----------
RRH Row:2 / Replicate Id: 323092490 / Table: stores9@informix.customer / DbOp:Insert
RRD 129||BBBBB
----------
RRH Row:3 / Replicate Id: 323092490 / Table: stores9@informix.customer / DbOp:Insert
RRD 130||BBBBB
----------
```

```
RRH Row:4 / Replicate Id: 323092490 / Table: stores9@informix.customer / DbOp:Insert
RRD 131||CCCCC
----------
RRH Row:5 / Replicate Id: 323092490 / Table: stores9@informix.customer / DbOp:Insert
RRD 132||CCCCC
```

5.6 Starting Enterprise Replication

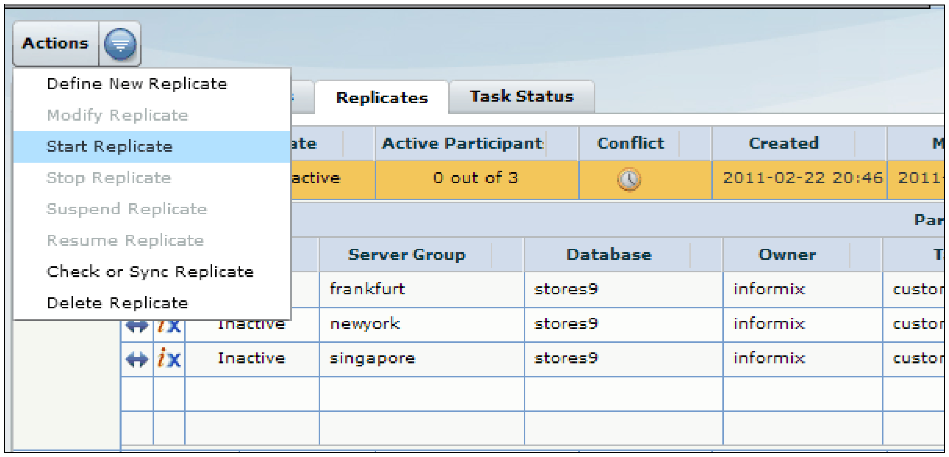
When a replicate is defined, it is *not* started by default. In order for a replicate to actually transfer data, it must be started.

Replicates may also be suspended or stopped. A stopped replication server does not evaluate data for replication or accumulate changes to be sent later. There is no replication activity. A suspended server does continue to evaluate data and collect what should be sent to other servers. When a suspended server is resumed, then the accumulated data is sent. When a stopped server is restarted, everything since the server was stopped is re-evaluated and sent as appropriate.

5.6.1 Starting Enterprise Replication using OAT

The process of starting a replicate from OAT is simple.

First, select the newly defined replicate raadr in the list of replicates displayed in the window under the Replicates tab. Click **Actions** and select **Start Replicate**, as shown in Figure 5-10.

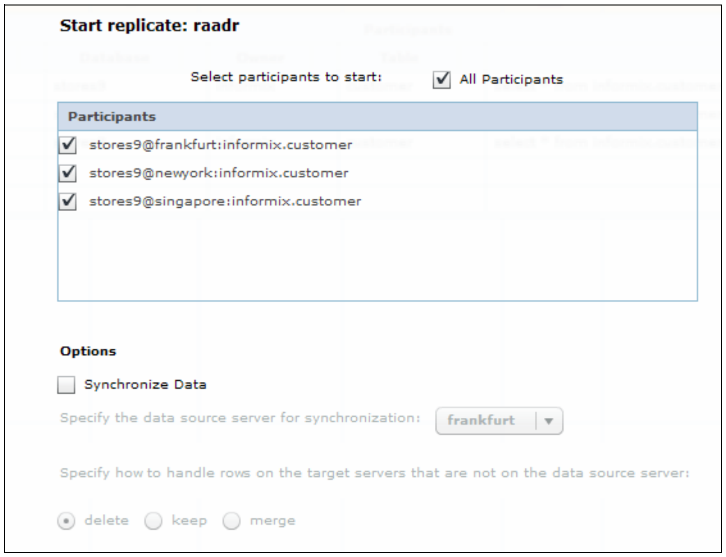


ate	Active Participant	Conflict	Created	M
active	0 out of 3	🕓	2011-02-22 20:46	2011

	Server Group	Database	Owner	T
	frankfurt	stores9	informix	custor
Inactive	newyork	stores9	informix	custor
Inactive	singapore	stores9	informix	custor

Figure 5-10 Start Replicate

You are now taken to the page showing the list of participants and other options. We select all the participants by selecting **All Participants** and keep the defaults for other options (Figure 5-11).



Start replicate: raadr

Select participants to start: [✓] All Participants

Participants
[✓] stores9@frankfurt:informix.customer
[✓] stores9@newyork:informix.customer
[✓] stores9@singapore:informix.customer

Options

[ ] Synchronize Data

Specify the data source server for synchronization: frankfurt

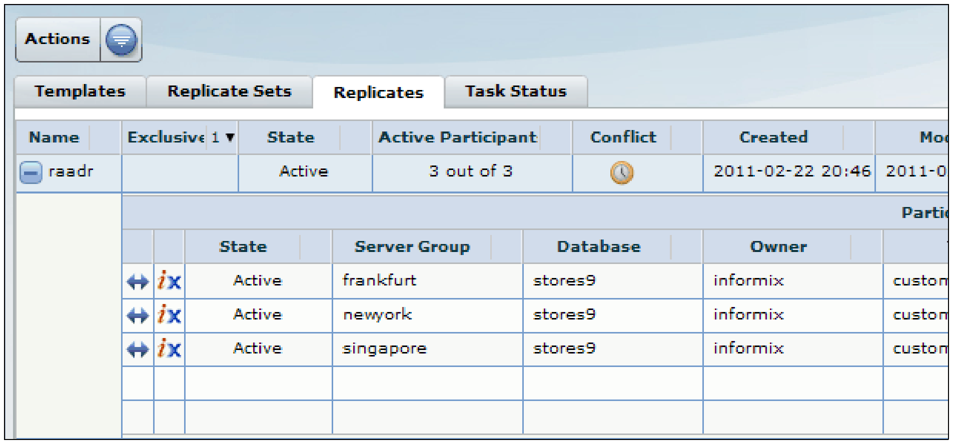
Specify how to handle rows on the target servers that are not on the data source server:

[● delete] keep merge

Figure 5-11 Start replicate options

Click **OK** to proceed with starting the replicate raadr. A window opens and shows the status of start replicate request. Click **Close** to return back to the window under the Replicates tab.

Figure 5-12 shows that the replicate raadr is in the Active state now.



Name	Exclusive 1 ▼	State	Active Participant	Conflict	Created	Mo
raadr		Active	3 out of 3		2011-02-22 20:46	2011-0

Parti

		State	Server Group	Database	Owner	
		Active	frankfurt	stores9	informix	custon
		Active	newyork	stores9	informix	custon
		Active	singapore	stores9	informix	custon

Figure 5-12 Replicate in Active state

At this time, the data is synchronized among the servers, and after that, changes are propagated according to the replicate definitions.

In our sample infrastructure, independent from the type of replicate and the data flow topology, after the replicate on the newyork server is started, the data changes are distributed to the targets as defined.

5.6.2 Starting Enterprise Replication using cdr

You can use the **`cdr start replicate`** command to start a newly defined replicate in an inactive state. Example 5-37 shows a replicate in an Inactive state, when listed using the **`cdr list replicate`** command.

Example 5-37 Replicate in an Inactive state

```
cdr list replicate raadr

#output
DEFINED REPLICATES ATTRIBUTES
------------------------------
REPLICATE:         raadr
STATE:             Inactive ON:newyork
CONFLICT:          Timestamp
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       stores9:informix.customer
OPTIONS:           row,fullrow
REPLID:            323092521 / 0x13420029
REPLMODE:          PRIMARY  ON:newyork
APPLY-AS:          INFORMIX ON:newyork
```

After starting the replicate with the **`cdr start replicate`** command, list the replicate to confirm that it is in the *Active* state now as shown in Example 5-38.

Example 5-38 Replicate in an Active state

```
cdr list replicate raadr

DEFINED REPLICATES ATTRIBUTES
------------------------------
REPLICATE:         raadr
STATE:             Active ON:newyork
CONFLICT:          Timestamp
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       stores9:informix.customer
OPTIONS:           row,fullrow
REPLID:            323092521 / 0x13420029
```

```
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork
```

5.7 Templates

Although Enterprise Replication supports a significant degree of functionality, empirical evidence indicates that more than 80 percent of the time, only 20 percent of the functionality is used. In most cases, the entire table is replicated, not just some subset of rows or columns. Also, more often than not, the same set of functionality is used for multiple tables within the database.

By defining a template, you can specify a set of tables and the characteristics to be used to propagate data changes. You can define the conflict resolution rules and the scope attributes, and define other attributes, such as ATS and RIS spooling, whether or not to fire triggers with the application, or whether or not to ignore deletes. During the definition of the template, you also obtain metadata that is used to describe the columns of the tables that make up the template. The replicates defined by the template will be defined as *mastered replicates* using this metadata.

After defining the template, you *realize* the template on the servers that are involved in replication. In effect, this means that the servers are added to the replicate definitions. The metadata that was obtained during the template definition is used to ensure that the tables match the expected attributes. Optionally, the table is created from the metadata during the template realization.

5.7.1 Terms and conditions

Templates cannot be used for all or for complex replication schemes due to some limits. A template applies to a single database. If multiple databases need to be replicated, then multiple templates are required.

Templates cannot be altered after they are defined. Any changes require deleting and re-creating the template.

Templates allow only full rows to be replicated. If partial rows are to be replicated, then that setup must be done using the more basic commands (such as **`cdr define replicate`**).

Templates do not allow time-based replication, such as at a specific time of day or at a specific interval. Only immediate data transfer is allowed. Templates also do not allow selecting which rows to be replicated to which target server.

5.7.2 Defining a template

Even with all of these restrictions, templates are a convenient way to set up most replication environments. In this section, we demonstrate how to set up replication on the stores_demo database using templates. There are three instances and the stores_demo database is created on two of the servers, but not the third server, as shown in Figure 5-13.

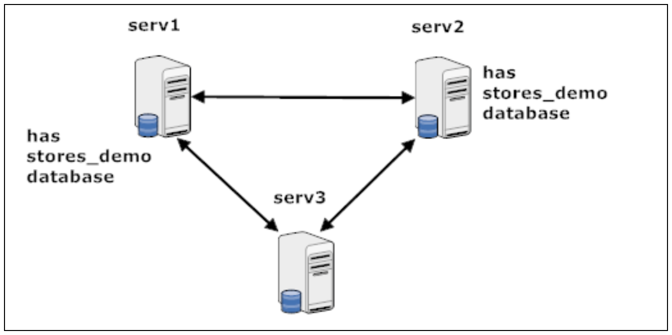


Figure 5-13 Setup for template

Using OAT to define a template

To define a template using OAT, complete the following steps:

1. Connect to a server:

   During the definition of the template, you must get the metadata that describes the tables that are included in the template. Therefore, the first step in defining a template is to attach to a server containing the tables that you want to include in the template. In our example, both serv1 and serv2 already contain the stores_demo database, so we select to connect to serv1. In OAT, select the server from the server drop-down menu that is found at the upper right part of the OAT window shown in Figure 5-14.

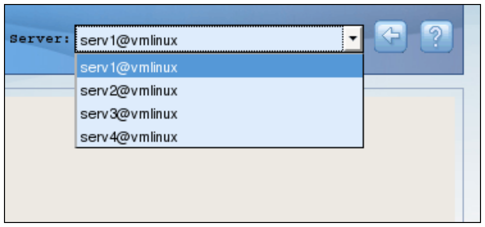
   

   Figure 5-14 Selecting a server in OAT

2. Navigate to the template definition window and select **Replication** → **Replicates**.

3. A new window opens that has four tabs labeled “Templates”, “Replicate Sets”, “Replicates”, and “Task Status”. Select **Templates**, as shown in Figure 5-15.

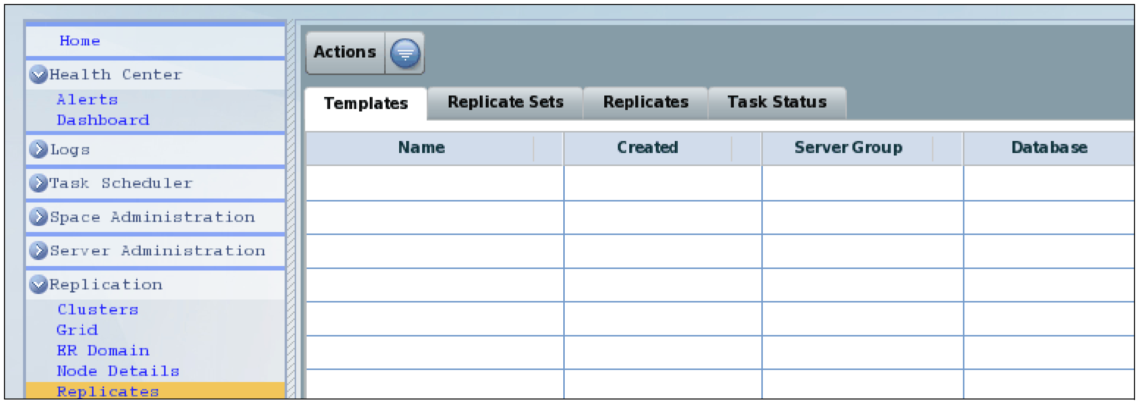


Figure 5-15 Selecting the Templates tab

4. Click the **Actions** drop-down menu and select **Define New Template**. The template definition window opens, as shown in Figure 5-16.

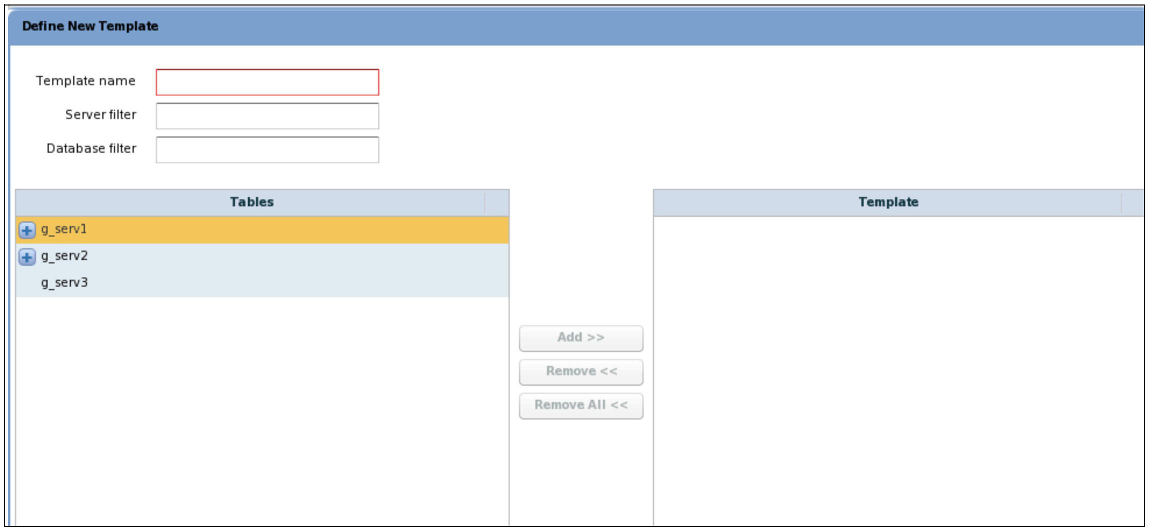
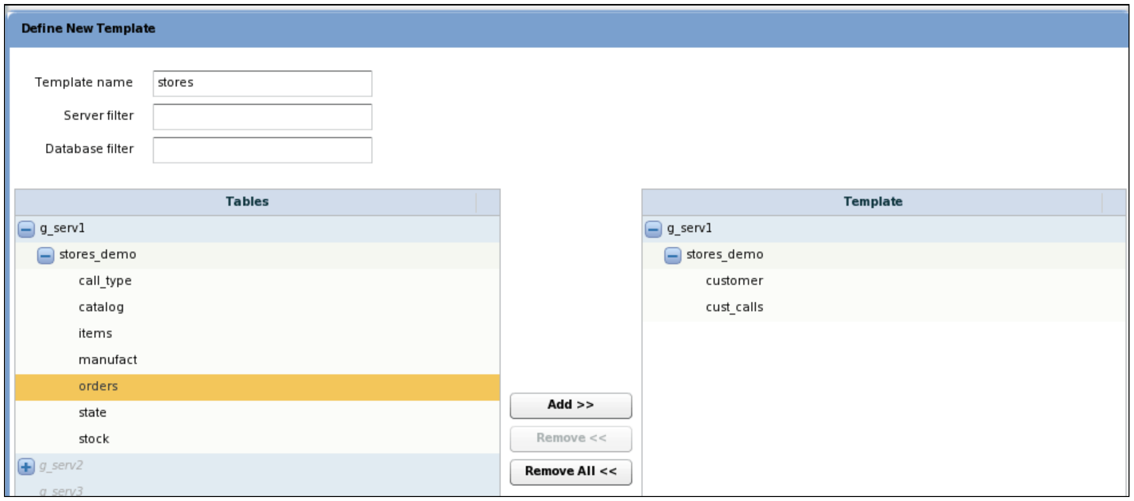


Figure 5-16 Initial template definition window

5. In the Define New Template window, specify a name for the template. In addition to naming the template, because a template is a specialized replicate set, this name is used to name that replicate set. In our case, we name the template “stores”.

To add a table to the template, click the **+** expansion button in front of the server (g_serv1) to get a list of databases that are candidates for replication. Then click the **+** expansion button in front of the database to get a list of the individual tables in the database that can be replicated. Highlight the individual table and move it into the template list, as shown in Figure 5-17.



Template name: stores
Server filter:
Database filter:

Tables:
- g_serv1
  - stores_demo
    - call_type
    - catalog
    - items
    - manufact
    - orders
    - state
    - stock
- g_serv2
- g_serv3

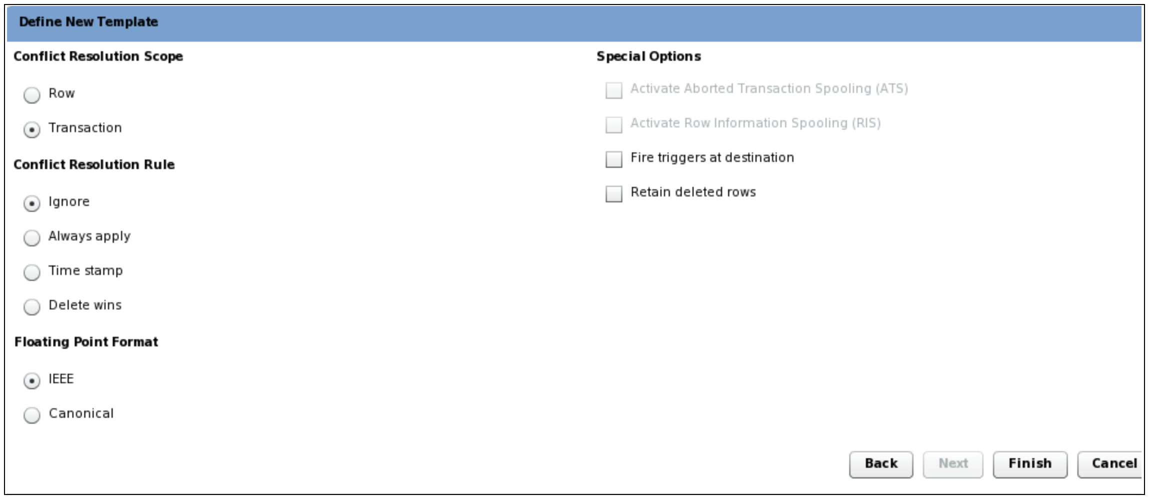
Template:
- g_serv1
  - stores_demo
    - customer
    - cust_calls

Figure 5-17 Selecting the tables to be included in the template

Optionally, you can highlight the stores_demo database and simply add all of the tables in the stores_demo database into the template list.

The "Server filter" and "Database filter" fields can be used to reduce the number of elements in the selection list.

In our example, we select all tables and then click **Next** to take us to the options window. The only thing that we change on this window is to select **Always apply** under the Conflict Resolution Rule, as shown in Figure 5-18.


Define New Template

Conflict Resolution Scope

- Row
- [✓ Transaction]

Conflict Resolution Rule

- [✓ Ignore]
- Always apply
- Time stamp
- Delete wins

Floating Point Format

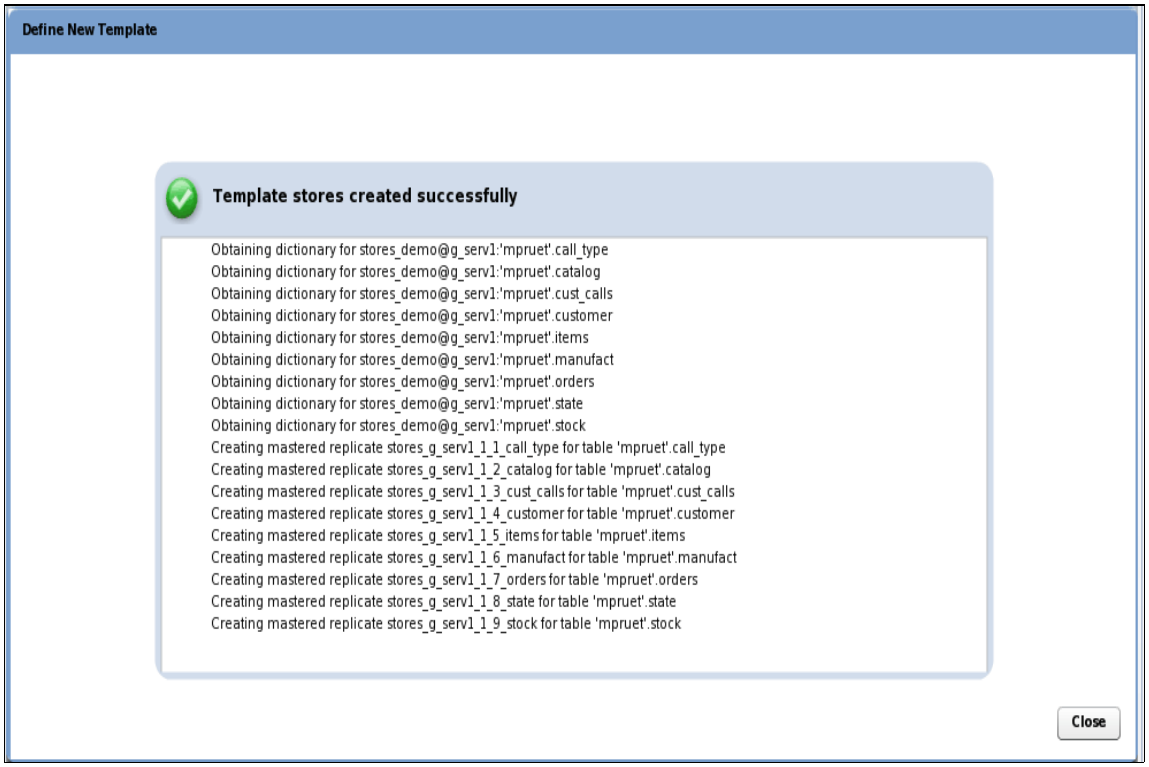
- [✓ IEEE]
- Canonical

Special Options

- [ ] Activate Aborted Transaction Spooling (ATS)
- [ ] Activate Row Information Spooling (RIS)
- [ ] Fire triggers at destination
- [ ] Retain deleted rows

Figure 5-18 Template options

6. Click **Finish** to define the template. Figure 5-19 shows the final window that describes the steps that the define template process performed.

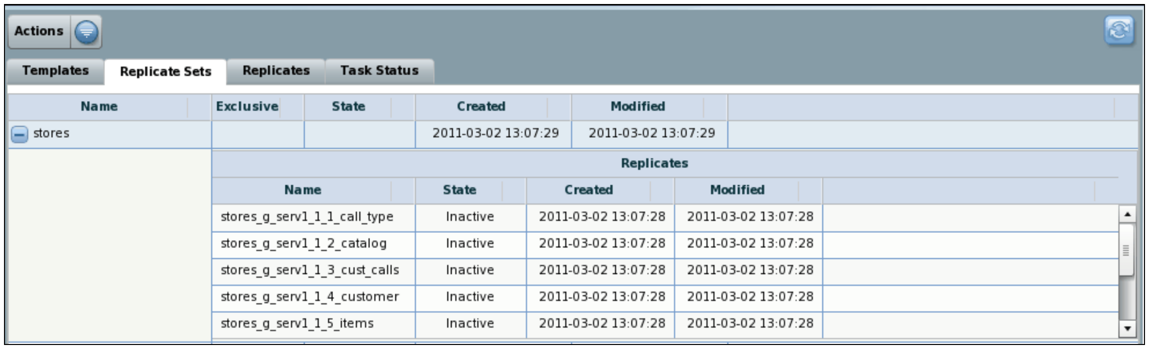


Template stores created successfully

```
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.call_type
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.catalog
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.cust_calls
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.customer
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.items
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.manufact
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.orders
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.state
Obtaining dictionary for stores_demo@g_serv1:'mpruet'.stock
Creating mastered replicate stores_g_serv1_1_1_call_type for table 'mpruet'.call_type
Creating mastered replicate stores_g_serv1_1_2_catalog for table 'mpruet'.catalog
Creating mastered replicate stores_g_serv1_1_3_cust_calls for table 'mpruet'.cust_calls
Creating mastered replicate stores_g_serv1_1_4_customer for table 'mpruet'.customer
Creating mastered replicate stores_g_serv1_1_5_items for table 'mpruet'.items
Creating mastered replicate stores_g_serv1_1_6_manufact for table 'mpruet'.manufact
Creating mastered replicate stores_g_serv1_1_7_orders for table 'mpruet'.orders
Creating mastered replicate stores_g_serv1_1_8_state for table 'mpruet'.state
Creating mastered replicate stores_g_serv1_1_9_stock for table 'mpruet'.stock
```

Figure 5-19 Work performed during the definition of a template

7. After clicking **Close**, you should return to the basic template window and the stores template now appears. Also, if you go into the Replicate Sets tab, you should see that there is also a stores replicate set. This is because a template is a special version of a replicate set. You can see the names of the replicates that are being used for each of the tables from the replicate set window, as shown in Figure 5-20.



Name	Exclusive	State	Created	Modified
stores			2011-03-02 13:07:29	2011-03-02 13:07:29

Replicates

Name	State	Created	Modified
stores_g_serv1_1_1_call_type	Inactive	2011-03-02 13:07:28	2011-03-02 13:07:28
stores_g_serv1_1_2_catalog	Inactive	2011-03-02 13:07:28	2011-03-02 13:07:28
stores_g_serv1_1_3_cust_calls	Inactive	2011-03-02 13:07:28	2011-03-02 13:07:28
stores_g_serv1_1_4_customer	Inactive	2011-03-02 13:07:28	2011-03-02 13:07:28
stores_g_serv1_1_5_items	Inactive	2011-03-02 13:07:28	2011-03-02 13:07:28

Figure 5-20 Replicates created as part of the store template

Using the cdr utility to define a template

You also can use the cdr utility to define the template. In fact, OAT indirectly uses the cdr utility to perform the definition of the template. Example 5-39 shows the cdr utility command to generate the stores template.

Example 5-39 cdr command to generate stores template

```
$cdr define template --connect=g_cdr1 --master=g_cdr1 stores \
    --conflict=always --scope=transaction --database=stores_demo \
    call_type catalog cust_calls customer items manufact orders state stock
```

It would also be possible to include all of the tables in the database in the template by using the --all option in place of the list of tables. But that can only be done if all of the tables can be replicated. For example, you cannot include a table that has no primary key.

An alternate method would be to create a file containing a list of tables that are included in the template. For example, if you have a file called stores_tab_list, then you can define the template, as shown in Example 5-40.

Example 5-40 Using a file to contain the list of tables for the define template

```
$cat stores_tab_list
call_type
catalog
cust_calls
```

```
customer
items
manufact
orders
state
stock

$cdr define template --connect=g_cdr1 --master_g_cdr1 stores \
    --conflict=always --scope=transaction --database=stores_demo \
    --file=stores_tab_list
```

Using the cdr list template

Example 5-41 shows how to use the cdr utility to display the replicates generated as part of a define template.

Example 5-41 cdr list template

```
$cdr list template stores
TEMPLATE:    stores
TEMPLATE ID: 65539
SERVER:      g_serv1
DATABASE:    stores_demo
REPLICATE:   stores_g_serv1_1_1_call_type
OWNER:       mpruet
TABLE:       call_type

TEMPLATE:    stores
TEMPLATE ID: 65539
SERVER:      g_serv1
DATABASE:    stores_demo
REPLICATE:   stores_g_serv1_1_2_catalog
OWNER:       mpruet
TABLE:       catalog

TEMPLATE:    stores
TEMPLATE ID: 65539
SERVER:      g_serv1
DATABASE:    stores_demo
REPLICATE:   stores_g_serv1_1_3_cust_calls
OWNER:       mpruet
TABLE:       cust_calls
...
```

Summary of defining templates

Here is a summary of defining templates:

- The template options are applied to all the tables. If your requirements do not allow that action, then multiple templates are required. If some tables require options different from the rest, those tables cannot all be in one template.
- A *mastered replicate* is defined, which means the replicate on each system where the template is realized *will* use the definition on HQ where the template is defined.
- Defining a template creates empty replicates without any participants. A template is just the specification or metadata used to operate on other systems when the template is realized.

5.7.3 Realizing the template

When you define a template, you end up with an empty replicate for each of the tables included in the template, which means that the replicate is not associated with any server. To actually replicate any data within the template, you must first associate that template with servers. This process is referred to as *realizing the template*.

When realizing a template, a number of things happen, which include:

1. Adding one or more participants to the replicate definition.
2. Creating tables, if necessary.
3. Ensuring that the tables on the various servers match the attributes of the template definition.
4. Populating the tables or synchronizing the data with the master data.
5. Starting the replicate set on the realized server(s).

Note: If a database is going to be created and populated, be sure that there is enough room for it. If not, the command fails.

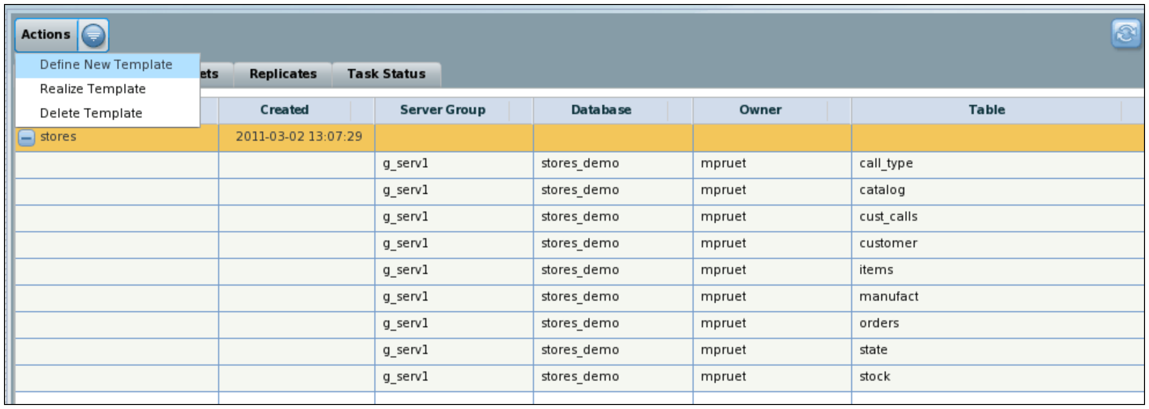
The process of realizing a template includes creating the database tables if they do not exist. In that case, there are options in the command to specify the dbspace in which the tables should reside (but the command does not create that dbspace) and the owner of the tables (the default being the informix user).

If the template realization includes the synchronization of date, then the template must be realized first on the server that is to be the sync server for the other targets. That is the first command in the example, and it must be first. If not, then the later commands fail because the source for the data is not known.

The template is realized on the remaining targets. In the example, there is only one target, but there could be more. In this example, the tables exist and hold data on the sync server but do not exist on the target database server. They are automatically created and populated from the sync server as part of realizing the template. Thus, the first command does not need to create any tables, but the second one does.

Using OAT to realize the template

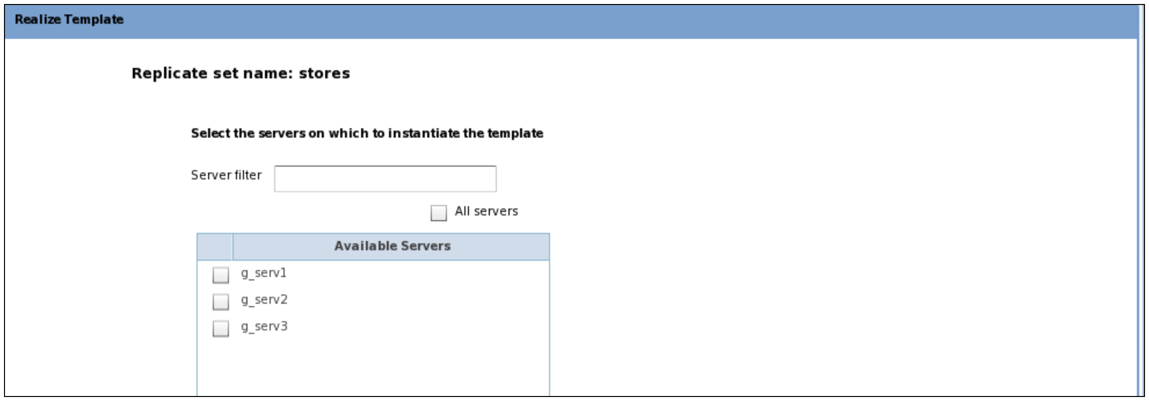
First, navigate to the template window by selecting **Replication** → **Replicates**. Select the **Templates** tab to open the template window. Highlight the template that you want to realize; in our case, this is the "stores" template. Expand the Actions drop-down list by clicking the down arrow. The window should look like Figure 5-21.



	Created	Server Group	Database	Owner	Table
stores	2011-03-02 13:07:29				
		g_serv1	stores_demo	mpruet	call_type
		g_serv1	stores_demo	mpruet	catalog
		g_serv1	stores_demo	mpruet	cust_calls
		g_serv1	stores_demo	mpruet	customer
		g_serv1	stores_demo	mpruet	items
		g_serv1	stores_demo	mpruet	manufact
		g_serv1	stores_demo	mpruet	orders
		g_serv1	stores_demo	mpruet	state
		g_serv1	stores_demo	mpruet	stock

Figure 5-21 Starting to realize the stores template

Select **Realize Template** and the Realize Template window opens (Figure 5-22).


Realize Template

Replicate set name: stores

Select the servers on which to instantiate the template

Server filter

☐ All servers

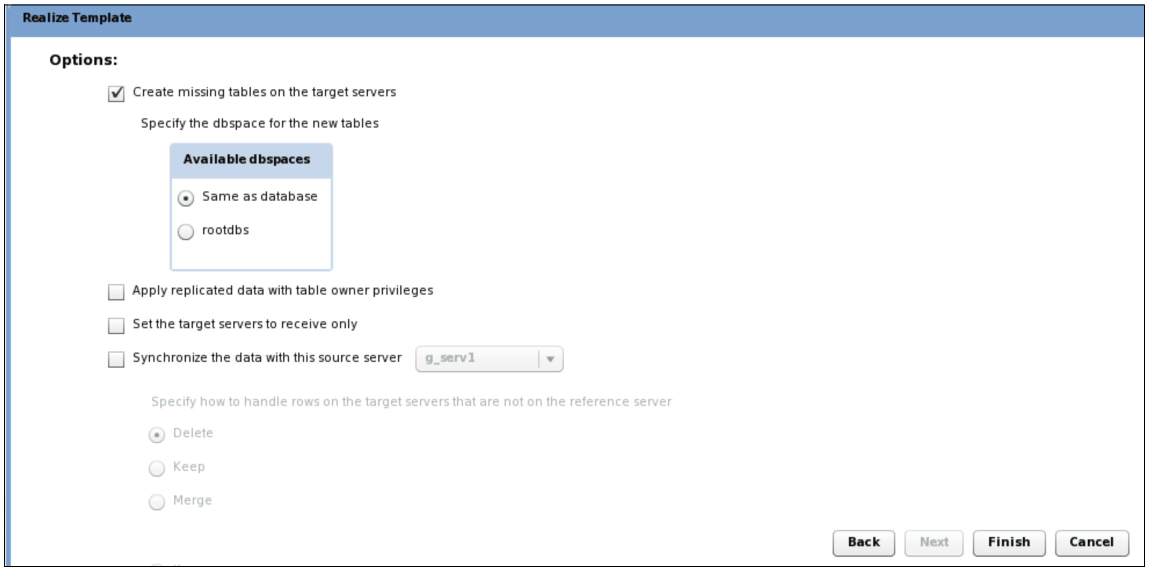
	Available Servers
☐	g_serv1
☐	g_serv2
☐	g_serv3

Figure 5-22 Selecting the servers for the template realization

Select your target server for replicate. In our example, we select serv1 and serv2, which already have the stores_demo database. Clicking **Next** leads to the options page for the replicate realization (Figure 5-23).

The options include:

- Options to create any tables that do not exist on the node. If the table does not exist, then the template metadata is used to create the table.
- An option to specify that the apply will be done as the table owner. If not selected, then the apply is performed as the informix user.
- Options to make the servers being realized as receive only.
- An option to perform a data synchronization as part of the realization


Realize Template

Options:

[✓] Create missing tables on the target servers

Specify the dbspace for the new tables

Available dbspaces

(●) Same as database

( ) rootdbs

[ ] Apply replicated data with table owner privileges

[ ] Set the target servers to receive only

[ ] Synchronize the data with this source server g_serv1

Specify how to handle rows on the target servers that are not on the reference server

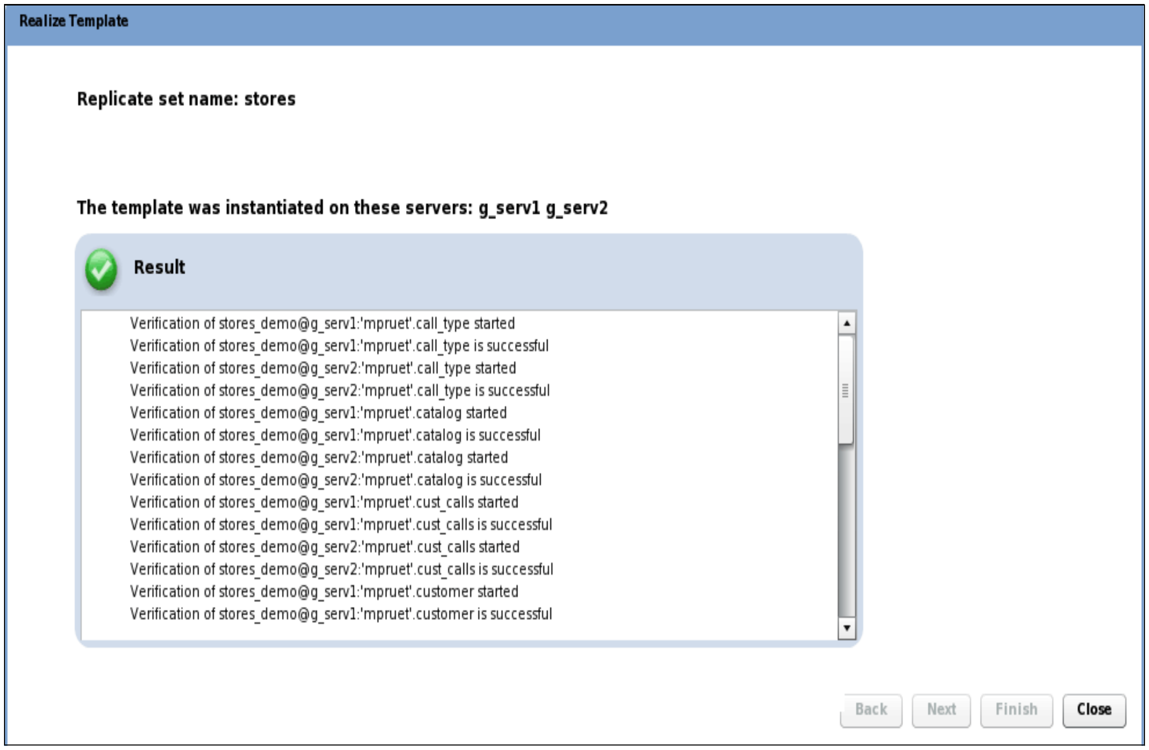
(●) Delete

( ) Keep

( ) Merge

Figure 5-23 Template realization options window

The two servers that we are realizing already have the stores_demo database installed, so there is no reason to select anything on this page. We clear the first check box to create missing tables. Click **Finish** to realize the stores template on the servers serv1 and serv2 (Figure 5-24).



Replicate set name: stores

The template was instantiated on these servers: g_serv1 g_serv2

Result

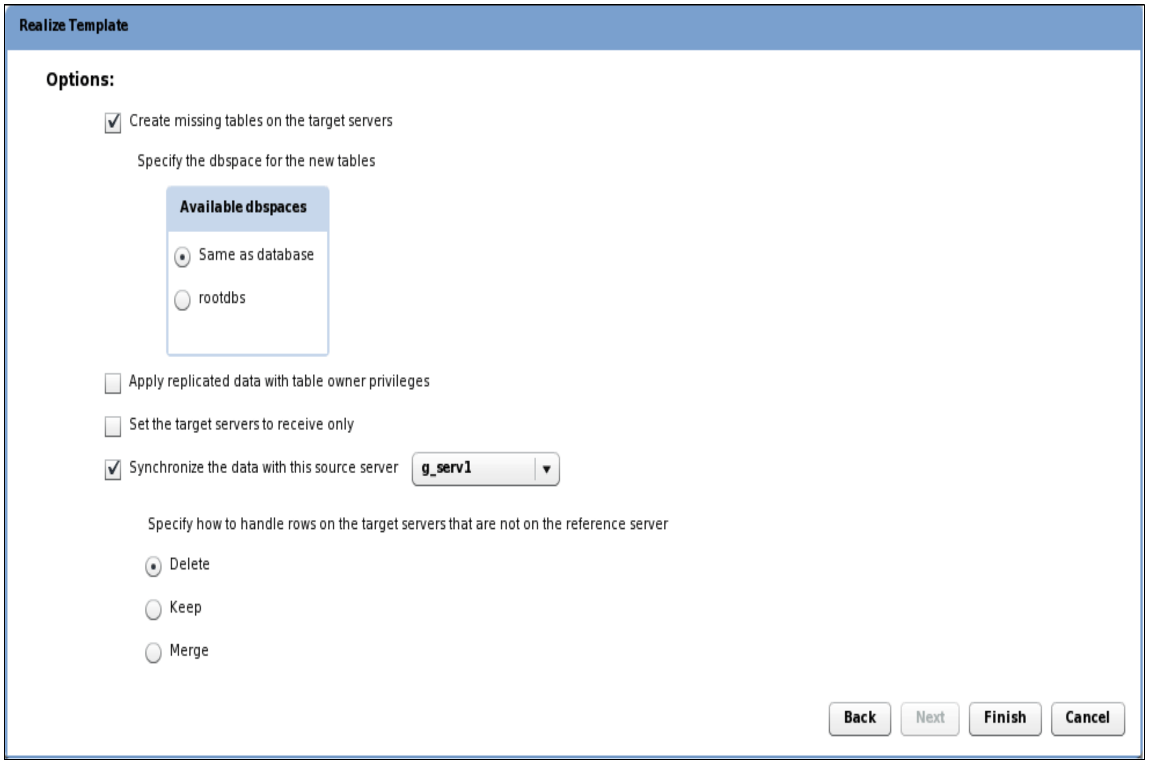
```
Verification of stores_demo@g_serv1:'mpruet'.call_type started
Verification of stores_demo@g_serv1:'mpruet'.call_type is successful
Verification of stores_demo@g_serv2:'mpruet'.call_type started
Verification of stores_demo@g_serv2:'mpruet'.call_type is successful
Verification of stores_demo@g_serv1:'mpruet'.catalog started
Verification of stores_demo@g_serv1:'mpruet'.catalog is successful
Verification of stores_demo@g_serv2:'mpruet'.catalog started
Verification of stores_demo@g_serv2:'mpruet'.catalog is successful
Verification of stores_demo@g_serv1:'mpruet'.cust_calls started
Verification of stores_demo@g_serv1:'mpruet'.cust_calls is successful
Verification of stores_demo@g_serv2:'mpruet'.cust_calls started
Verification of stores_demo@g_serv2:'mpruet'.cust_calls is successful
Verification of stores_demo@g_serv1:'mpruet'.customer started
Verification of stores_demo@g_serv1:'mpruet'.customer is successful
```

Figure 5-24 Result of template realization

Realizing the template with table creation

In our example, the stores_demo database is not currently installed on serv3. Therefore, when we realize the stores template on serv3, we must specify that we want to automatically create any missing tables and also perform an initial synchronization as part of the realization. We go through most of the same steps as before by first navigating to the template page (select **Replication** → **Replicates**) and then selecting the templates tab. After we highlight the stores template and click **Realize Template** under the **Actions** drop-down menu, we are ready to realize the stores template on serv3. We continue by selecting the g_serv3 entry and clicking **Next**.

Because the stores_demo does not exist on serv3, we must select **Create missing tables on the target servers**. Also, we have to specify that we should synchronize the data as part of the realization, as shown in Figure 5-25. At this point, we click **Finish** to realize the template.


Realize Template

Options:

[✓] Create missing tables on the target servers

Specify the dbspace for the new tables

Available dbspaces

[● Same as database] ○ rootdbs

[ ] Apply replicated data with table owner privileges

[ ] Set the target servers to receive only

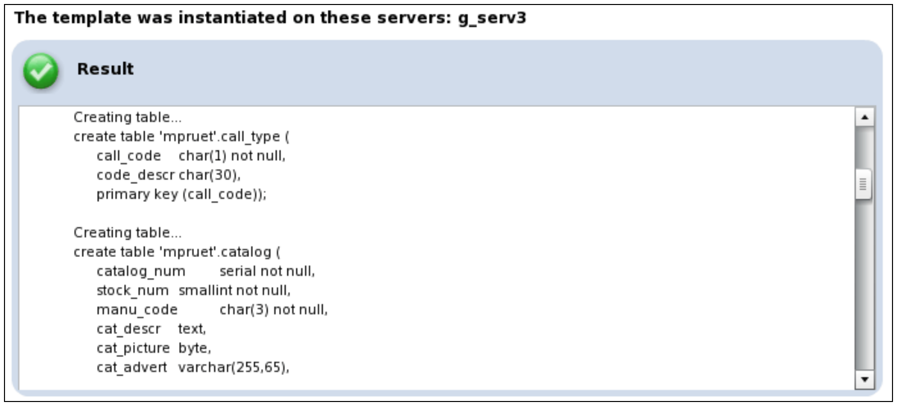
[✓] Synchronize the data with this source server: g_serv1

Specify how to handle rows on the target servers that are not on the reference server

[● Delete] ○ Keep ○ Merge

Figure 5-25 Realization of the stores template on serv3

The output of the realize template results also include the SQL statements for the tables that are generated (Figure 5-26).

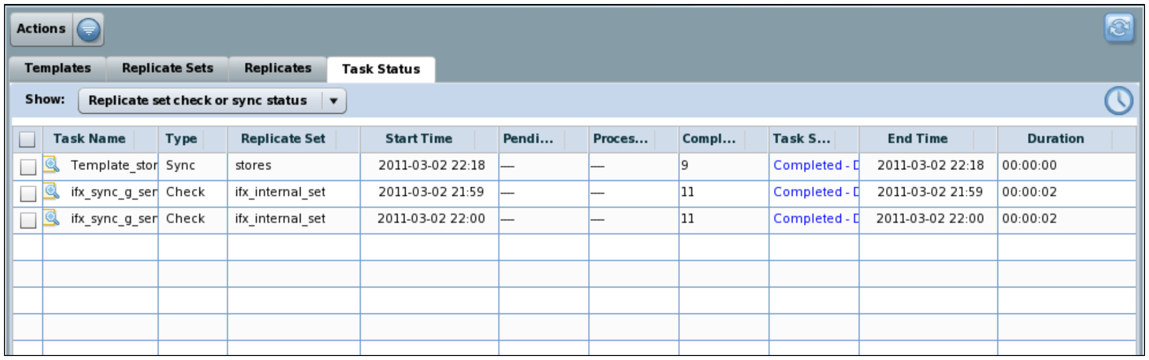

The template was instantiated on these servers: g_serv3

Result

```
Creating table...
create table 'mpruet'.call_type (
    call_code    char(1) not null,
    code_descr char(30),
    primary key (call_code));

Creating table...
create table 'mpruet'.catalog (
    catalog_num       serial not null,
    stock_num  smallint not null,
    manu_code          char(3) not null,
    cat_descr    text,
    cat_picture  byte,
    cat_advert   varchar(255,65),
```

Figure 5-26 Output of template realization

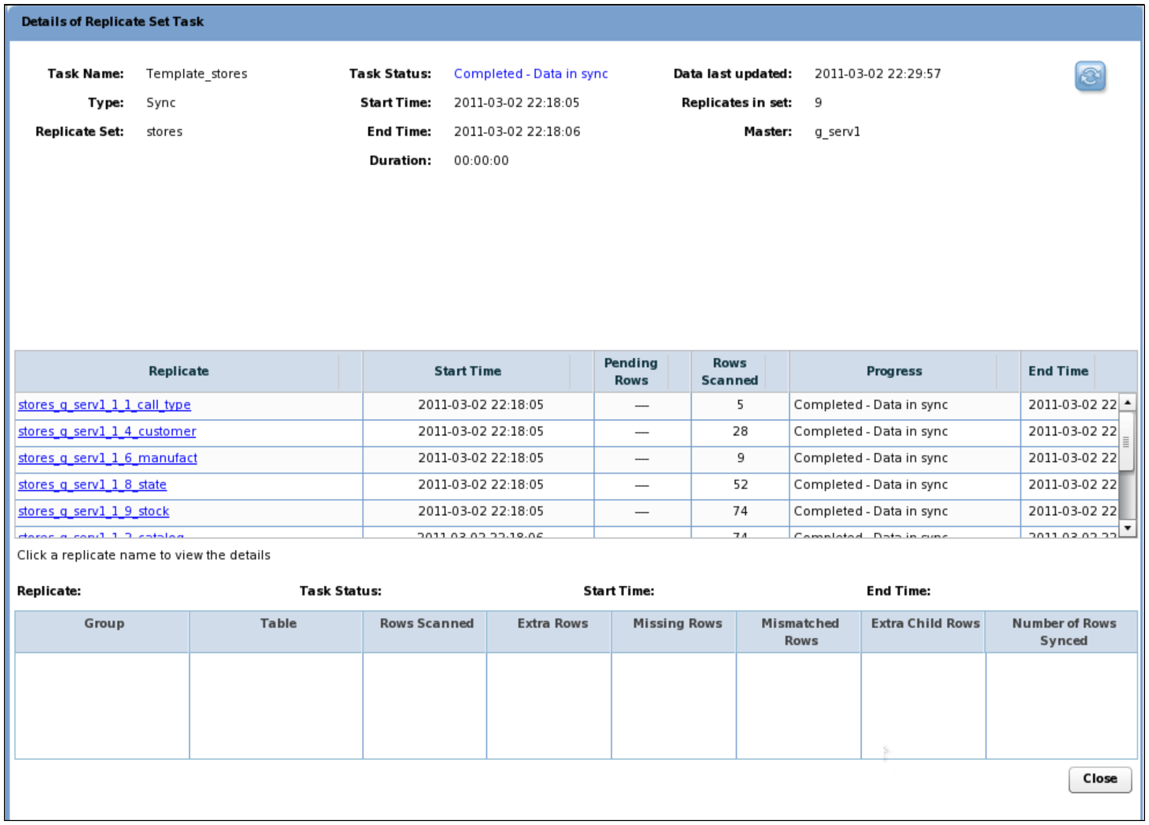
We can verify that the synchronization is performed by navigating to the **Task Status** tab and then clicking **Replicate set check or sync status**. We see a window similar to Figure 5-27.



	Task Name	Type	Replicate Set	Start Time	Pendi...	Proces...	Compl...	Task S...	End Time	Duration
	Template_stor	Sync	stores	2011-03-02 22:18	—	—	9	Completed - D	2011-03-02 22:18	00:00:00
	ifx_sync_g_ser	Check	ifx_internal_set	2011-03-02 21:59	—	—	11	Completed - D	2011-03-02 21:59	00:00:02
	ifx_sync_g_ser	Check	ifx_internal_set	2011-03-02 22:00	—	—	11	Completed - D	2011-03-02 22:00	00:00:02

Figure 5-27 Replicate set check and sync status

We click the icon and see the results of the task. Because we are realizing the stores template, the sync would be the store replicate set. If we click the first task, we see the window shown in Figure 5-28, which indicates that the sync process was successful.


Details of Replicate Set Task

Task Name:	Template_stores	**Task Status:**	Completed - Data in sync	**Data last updated:**	2011-03-02 22:29:57
Type:	Sync	**Start Time:**	2011-03-02 22:18:05	**Replicates in set:**	9
Replicate Set:	stores	**End Time:**	2011-03-02 22:18:06	**Master:**	g_serv1
		Duration:	00:00:00		

Replicate	Start Time	Pending Rows	Rows Scanned	Progress	End Time
stores_g_serv1_1_1_call_type	2011-03-02 22:18:05	—	5	Completed - Data in sync	2011-03-02 22
stores_g_serv1_1_4_customer	2011-03-02 22:18:05	—	28	Completed - Data in sync	2011-03-02 22
stores_g_serv1_1_6_manufact	2011-03-02 22:18:05	—	9	Completed - Data in sync	2011-03-02 22
stores_g_serv1_1_8_state	2011-03-02 22:18:05	—	52	Completed - Data in sync	2011-03-02 22
stores_g_serv1_1_9_stock	2011-03-02 22:18:05	—	74	Completed - Data in sync	2011-03-02 22

Click a replicate name to view the details

Replicate: **Task Status:** **Start Time:** **End Time:**

Group	Table	Rows Scanned	Extra Rows	Missing Rows	Mismatched Rows	Extra Child Rows	Number of Rows Synced

Figure 5-28 Results of initial sync as part of realize template

Realizing a template using the cdr utility

You can also realize a template by using the cdr utility. Example 5-42 shows the help output for the command from running **`cdr realize template -h`**.

Example 5-42 cdr realize template help panel

```
cdr realize template -x
usage: cdr realize template TemplateName [-c server] [-D dbspace] [-ouvt]
                   [-S server] [-e delete|keep|merge] "db@serv" ...
 -c server --connect=server  connect to server
 -D        --dbspace=<spacename> dbspace to be used to create tables.
 -u        --autocreate automatically create tables if they do not exists
 -v        --verify  verify the existing replicates using template definition
```

```
-t        --target realize template as a receive only server
-S server --syncdatasource=server server to be used as a data source for sync
-J        --memadjust=<size_to_adjust[K|M]
                         Amount of memory to allow cdr send queue to
                         expand to while the sync is executed.
-F        --foreground   Execute the sync as a forground task
-e        --extratargetrows=[keep|delete|merge] handle extra row during sync
-o        --applyasowner realize template as the owner of the table
```

The options are similar to the selections for the OAT except option --memadjust, which can be used to change the queue size while the initial sync is underway.

Example 5-43 shows the **cdr** command that is used to realize a template.

Example 5-43 Realize a template using cdr

```
cdr realize template --connect=serv1 stores g_serv1 g_serv2
Verification of stores_demo@g_serv1:'mpruet'.call_type started
Verification of stores_demo@g_serv1:'mpruet'.call_type is successful
Verification of stores_demo@g_serv2:'mpruet'.call_type started
Verification of stores_demo@g_serv2:'mpruet'.call_type is successful
Verification of stores_demo@g_serv1:'mpruet'.catalog started
Verification of stores_demo@g_serv1:'mpruet'.catalog is successful
Verification of stores_demo@g_serv2:'mpruet'.catalog started
Verification of stores_demo@g_serv2:'mpruet'.catalog is successful
Verification of stores_demo@g_serv1:'mpruet'.cust_calls started
Verification of stores_demo@g_serv1:'mpruet'.cust_calls is successful
Verification of stores_demo@g_serv2:'mpruet'.cust_calls started
Verification of stores_demo@g_serv2:'mpruet'.cust_calls is successful
Verification of stores_demo@g_serv1:'mpruet'.customer started
Verification of stores_demo@g_serv1:'mpruet'.customer is successful
Verification of stores_demo@g_serv2:'mpruet'.customer started
Verification of stores_demo@g_serv2:'mpruet'.customer is successful
Verification of stores_demo@g_serv1:'mpruet'.items started
Verification of stores_demo@g_serv1:'mpruet'.items is successful
Verification of stores_demo@g_serv2:'mpruet'.items started
Verification of stores_demo@g_serv2:'mpruet'.items is successful
Verification of stores_demo@g_serv1:'mpruet'.manufact started
Verification of stores_demo@g_serv1:'mpruet'.manufact is successful
Verification of stores_demo@g_serv2:'mpruet'.manufact started
Verification of stores_demo@g_serv2:'mpruet'.manufact is successful
Verification of stores_demo@g_serv1:'mpruet'.orders started
Verification of stores_demo@g_serv1:'mpruet'.orders is successful
Verification of stores_demo@g_serv2:'mpruet'.orders started
Verification of stores_demo@g_serv2:'mpruet'.orders is successful
Verification of stores_demo@g_serv1:'mpruet'.state started
Verification of stores_demo@g_serv1:'mpruet'.state is successful
Verification of stores_demo@g_serv2:'mpruet'.state started
Verification of stores_demo@g_serv2:'mpruet'.state is successful
Verification of stores_demo@g_serv1:'mpruet'.stock started
Verification of stores_demo@g_serv1:'mpruet'.stock is successful
Verification of stores_demo@g_serv2:'mpruet'.stock started
Verification of stores_demo@g_serv2:'mpruet'.stock is successful
```

When realizing the template on a server, you have to automatically create the tables and perform an initial synchronization. Example 5-44 shows how to complete this task on serv3.

Example 5-44 Realize a template with automatic create and initial sync

```
$cdr realize template stores --connect=g_serv1 --autocreate
--syncdatasource=g_serv1 --foreground g_serv3
Verification of stores_demo@g_serv3:'mpruet'.call_type started
Creating table...
create table 'mpruet'.call_type (
        call_code       char(1) not null,
        code_descr      char(30),
        primary key (call_code));

Verification of stores_demo@g_serv3:'mpruet'.call_type is successful
Verification of stores_demo@g_serv3:'mpruet'.catalog started
Creating table...
create table 'mpruet'.catalog (
        catalog_num     serial not null,
        stock_num       smallint not null,
        manu_code       char(3) not null,
        cat_descr       text,
        cat_picture     byte,
        cat_advert      varchar(255,65),
        primary key (catalog_num));

Verification of stores_demo@g_serv3:'mpruet'.catalog is successful
Verification of stores_demo@g_serv3:'mpruet'.cust_calls started
Creating table...
create table 'mpruet'.cust_calls (
        customer_num    integer not null,
        call_dtime      datetime year to minute not null,
        user_id char(32),
        call_code       char(1),
        call_descr      char(240),
        res_dtime       datetime year to minute,
        res_descr       char(240),
        primary key (customer_num, call_dtime));

Verification of stores_demo@g_serv3:'mpruet'.cust_calls is successful
Verification of stores_demo@g_serv3:'mpruet'.customer started
Creating table...
create table 'mpruet'.customer (
        customer_num    serial not null,
        fname   char(15),
        lname   char(15),
        company char(20),
        address1        char(20),
        address2        char(20),
        city    char(15),
        state   char(2),
        zipcode char(5),
        phone   char(18),
        primary key (customer_num));

Verification of stores_demo@g_serv3:'mpruet'.customer is successful
Verification of stores_demo@g_serv3:'mpruet'.items started
Creating table...
create table 'mpruet'.items (
        item_num        smallint not null,
        order_num       integer not null,
        stock_num       smallint not null,
        manu_code       char(3) not null,
        quantity        smallint,
        total_price     money(8,2),
        primary key (item_num, order_num));

Verification of stores_demo@g_serv3:'mpruet'.items is successful
Verification of stores_demo@g_serv3:'mpruet'.manufact started
Creating table...
create table 'mpruet'.manufact (
        manu_code       char(3) not null,
        manu_name       char(15),
        lead_time       interval day(3) to day,
        primary key (manu_code));

Verification of stores_demo@g_serv3:'mpruet'.manufact is successful
Verification of stores_demo@g_serv3:'mpruet'.orders started
Creating table...
create table 'mpruet'.orders (
        order_num       serial not null,
        order_date      date,
        customer_num    integer not null,
        ship_instruct   char(40),
        backlog char(1),
        po_num  char(10),
        ship_date       date,
        ship_weight     decimal(8, 2),
        ship_charge     money(6,2),
        paid_date       date,
        primary key (order_num));

Verification of stores_demo@g_serv3:'mpruet'.orders is successful
Verification of stores_demo@g_serv3:'mpruet'.state started
Creating table...
create table 'mpruet'.state (
        code    char(2) not null,
        sname   char(15),
        primary key (code));

Verification of stores_demo@g_serv3:'mpruet'.state is successful
Verification of stores_demo@g_serv3:'mpruet'.stock started
Creating table...
create table 'mpruet'.stock (
        stock_num       smallint not null,
        manu_code       char(3) not null,
        description     char(15),
        unit_price      money(6,2),
        unit    char(4),
        unit_descr      char(15),
        primary key (stock_num, manu_code));


Verification of stores_demo@g_serv3:'mpruet'.stock is successful
Mar 03 2011 09:55:14 ------   Table scan for stores_g_serv1_1_8_state start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_8_state

Ending synchronization scan for replicate
          stores_g_serv1_1_8_state
Ending synchronization scan for replicate
          stores_g_serv1_1_8_state
Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_8_state end   ---------


Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_4_customer start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_4_customer

Ending synchronization scan for replicate
          stores_g_serv1_1_4_customer
Ending synchronization scan for replicate
```

```
          stores_g_serv1_1_4_customer
Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_4_customer end   ---------


Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_6_manufact start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_6_manufact

Ending synchronization scan for replicate
          stores_g_serv1_1_6_manufact
Ending synchronization scan for replicate
          stores_g_serv1_1_6_manufact
Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_6_manufact end   ---------


Mar 03 2011 09:55:16 ------   Table scan for stores_g_serv1_1_1_call_type start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_1_call_type

Ending synchronization scan for replicate
          stores_g_serv1_1_1_call_type
Ending synchronization scan for replicate
          stores_g_serv1_1_1_call_type
Mar 03 2011 09:55:17 ------   Table scan for stores_g_serv1_1_1_call_type end   ---------


Mar 03 2011 09:55:17 ------   Table scan for stores_g_serv1_1_9_stock start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_9_stock

Ending synchronization scan for replicate
          stores_g_serv1_1_9_stock
Ending synchronization scan for replicate
          stores_g_serv1_1_9_stock
Mar 03 2011 09:55:18 ------   Table scan for stores_g_serv1_1_9_stock end   ---------


Mar 03 2011 09:55:18 ------   Table scan for stores_g_serv1_1_7_orders start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_7_orders

Ending synchronization scan for replicate
          stores_g_serv1_1_7_orders
Ending synchronization scan for replicate
          stores_g_serv1_1_7_orders
Mar 03 2011 09:55:18 ------   Table scan for stores_g_serv1_1_7_orders end   ---------


Mar 03 2011 09:55:19 ------   Table scan for stores_g_serv1_1_3_cust_calls start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_3_cust_calls

Ending synchronization scan for replicate
          stores_g_serv1_1_3_cust_calls
Ending synchronization scan for replicate
          stores_g_serv1_1_3_cust_calls
Mar 03 2011 09:55:19 ------   Table scan for stores_g_serv1_1_3_cust_calls end   ---------


Mar 03 2011 09:55:19 ------   Table scan for stores_g_serv1_1_2_catalog start  --------
Starting synchronization scan for replicate
          stores_g_serv1_1_2_catalog

Ending synchronization scan for replicate
          stores_g_serv1_1_2_catalog
Ending synchronization scan for replicate
          stores_g_serv1_1_2_catalog
```

```
Mar 03 2011 09:55:20 ------   Table scan for stores_g_serv1_1_2_catalog end   ---------


Mar 03 2011 09:55:20 ------   Table scan for stores_g_serv1_1_5_items start  --------
Starting synchronization scan for replicate
         stores_g_serv1_1_5_items

Ending synchronization scan for replicate
         stores_g_serv1_1_5_items
Ending synchronization scan for replicate
         stores_g_serv1_1_5_items
Mar 03 2011 09:55:20 ------   Table scan for stores_g_serv1_1_5_items end   ---------
```

Summary of the realizing template

Here is a summary of the realizing template:

- When realizing a template, attach a template definition to a server.
- Replication is not actually started until the template realization occurs.
- During the template realization, verify that the table schemas on the servers being realized match template metadata.
- If the table does not exist on the server being realized, a table can be created automatically.
- You can perform an initial synchronization as part of the template realization.

5.8 Other replication considerations

There are additional functionality that must be included in the discussion about replication.

5.8.1 Blocking and unblocking replication

You can block replication from a complete transaction by using *begin work without replication*. This functionality has been in place since Enterprise Replication was first released. When performing operations from a transaction that was started as begin work without replication, Informix flags the log records so that they will not be snooped.

With the release of 11.70, you also can turn off the replication flag from within the transaction by using the built-in function ifx_set_erstate(). If the parameter to the function call is the number 1 or the character string “on”, then replication is activated. If the parameter is the number 0 or the character string “off”, then replication is deactivated.

There is a similar built-in function that returns the current state of replication. The ifx_get_erstate() function takes no parameters and returns the integer 1 if replication is active and the integer 0 if it is not.

Suppose that we have a stored procedure that is executed from a trigger and the table to which the trigger involves is also replicated by Enterprise Replication. Also suppose that the replicate definition specifies that triggers are supposed to fire on the target servers. We might want the results of the stored procedure execution to be replicated. However, normally anything that is done as part of the ER application is not replicated, which puts us in the situation where we want to replicate some, but not all of, the changes that are performed by ER application. To make matters even more complicated, we sometimes execute the trigger through a user thread. We could implement the code in Example 5-45 to handle this issue.

Example 5-45 Turning on / off replication from within a transaction

```
create procedure my_procedure()
define orig_repstate integer;
-- save the current replication state
execute function ifx_get_erstate() into repstate;
-- turn on replication
execute procedure ifx_set_erstate('on');
--
-- Body of stored procedure
--
...
-- Now reset the replication state
execute procedure ifx_set_erstate(orig_repstate);
end procedure;
```

In this example, we:

1. Save the original replication state in a variable.
2. Turn on replication by executing **`ifx_set_erstate`**.
3. Reset the replication state back to what it was when we entered the procedure.

If the state originally had replication turned on, then turning it on within the stored procedure does no harm. Likewise, if it was originally off, then we will turn it back off when exiting the procedure.

5.8.2 Detecting whether a procedure is running by viewing the replication apply thread

Often we need to know if we are executing a stored procedure from a trigger that was fired from a replication apply thread. As an example, suppose that we want to execute a system call, but only if the trigger is being fired on the system where an insert was actually performed. We can use **`dbinfo`** to detect this situation.

The format is **`dbinfo('cdrsession')`**. If the return value is 1, then we are executing as part of Enterprise Replication apply. If the return is 0, then we are not. Example 5-46 demonstrates this situation.

Example 5-46 Using dbinfo to detect if a procedure is running from the ER apply

```
CREATE DATABASE cmpdb with log;

CREATE TABLE tab1(
    col1    integer primary key,
    col2    integer) with crcols lock mode row;

CREATE PROCEDURE postrow(insrow integer)
define cmd varchar(255);

if (dbinfo('cdrsession') = 0) then
    let cmd = "echo " || DBSERVERNAME || " ins row " ||
        insrow || " >> /tmp/inslog";
else
    let cmd = "echo " || DBSERVERNAME || " replicate row insert " ||
        insrow || " >> /tmp/inslog";
end if;
    system cmd;
end procedure;

CREATE TRIGGER trig1 INSERT on tab1
  REFERENCING new as nrow
  FOR EACH ROW (EXECUTE PROCEDURE postrow(nrow.col1));
```

In this example, we define the trigger to execute the postrow() stored procedure and the stored procedure behaves a bit differently depending on whether it is executed by the Enterprise Replication apply or by a user application. If it is executed by a user application, then it posts a message in `/tmp/inslog` identifying the operation as an insert. If it is executed by an Enterprise Replication apply, then Informix posts a different message. After inserting a couple of rows on serv1 and then one row on serv2, the combined results look like Example 5-47.

Example 5-47 Results of three inserts

```
serv1 ins row 1
serv2 replicate row insert 1
serv3 replicate row insert 1
serv1 ins row 2
serv3 replicate row insert 2
serv2 replicate row insert 2
serv2 ins row 3
serv1 replicate row insert 3
serv3 replicate row insert 3
```

5.8.3 Replicating user defined types and routines

User defined types (UDTs) and routines (UDRs) require special treatment.

First, user defined routines are not propagated. Changes must be transported and installed on the replication servers outside the replication processes. When a replication network is first set up, each server should have the same version of any UDR that affects any columns in any replicates. If changes are required, those changes must be applied separately on each replication server.

UDTs that are not opaque are replicated in the same way as the built-in data types. Opaque types can be replicated only if certain support functions are provided. Because the DBMS does not know the details of the opaque types, the streamwrite() and streamread() functions are required so the replication threads are able to read and write the data correctly. In addition, all the UDTs that will be replicated must be defined on each replication server before replication begins. The types (or changes to the types) are *not* replicated.

For additional information about this topic, see the article on IBM developerWorks® found at the following address:

http://www.ibm.com/developerworks/db2/zones/informix/library/techarticle/0208pincheira/0208pincheira.html

5.9 Maintaining a replication server

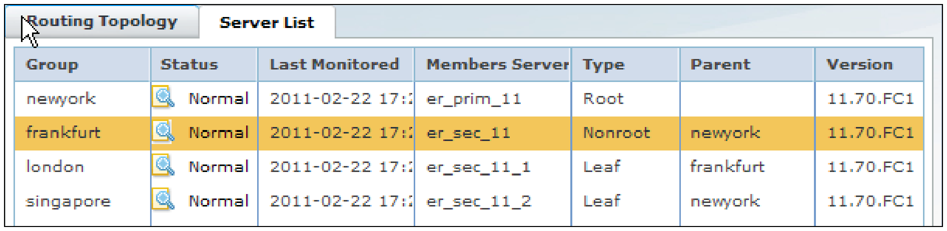
Due to changing business needs and requirements, replication server topologies require the addition, modification, and deletion of replication servers. Because replication servers are the replication gatekeepers, they also have be started, stopped, suspended, and resumed.

5.9.1 Finding replication servers in an ER domain

After a set of replication servers are set up in an ER domain in a specific topology, it is necessary to have a periodic snapshot of the entire ER domain. The **`cdr list server`** command provides the ability to list all the servers, along with the interconnection information about each of the global catalogs of an ER server. The command can be executed against any of the replication servers.

Note: Because the **`cdr list server`** command uses information in the global catalog of an ER server, the command can be executed against any of the root or non-root servers in a particular ER domain to list information about all the members of that ER domain.

Figure 5-29 shows that newyork is configured as the root node. The **`cdr list server`** command is executed to find or list the replication servers that have been defined. To perform this action in OAT, select **Replication** → **ER Domain** → **Server List**.



Group	Status	Last Monitored	Members Server	Type	Parent	Version
newyork	Normal	2011-02-22 17::	er_prim_11	Root		11.70.FC1
frankfurt	Normal	2011-02-22 17::	er_sec_11	Nonroot	newyork	11.70.FC1
london	Normal	2011-02-22 17::	er_sec_11_1	Leaf	frankfurt	11.70.FC1
singapore	Normal	2011-02-22 17::	er_sec_11_2	Leaf	newyork	11.70.FC1

Figure 5-29 Server list

5.9.2 Modifying a replication server

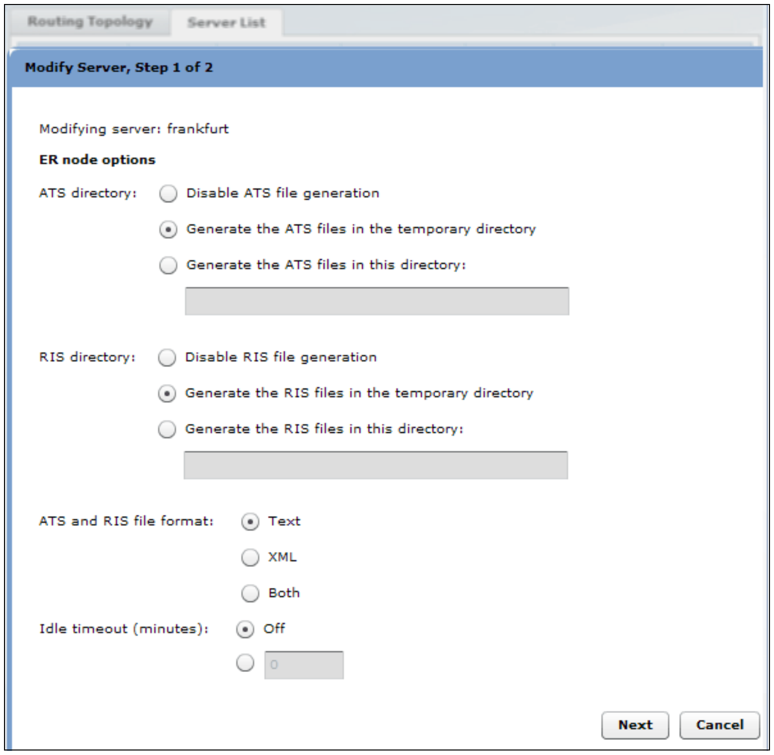
After there are replication servers defined in an ER domain, some of the attributes of each of the servers can be changed by using the **`cdr modify server`** command.

This **cdr modify server** command allows the modification of a replication server definition, but is limited to modifying only the idle timeout, location of the aborted transaction spooling (ATS) files, and location of the row information spooling (RIS) files, as well as changing the mode (to primary or read-only) of all replicates with ignore as the conflict rule.

Note: The **cdr modify server** command can be executed only on a root or non-root node of the ER domain. It cannot be run on a leaf node.

To modify a replication server using OAT, select **Replication** → **ER Domain** → **Server List** → **Frankfurt** → **Actions** → **Modify Server**.

Figure 5-30 and Figure 5-31 on page 182 show the OAT view of **cdr modify server**.



Modify Server, Step 1 of 2

Modifying server: frankfurt

ER node options

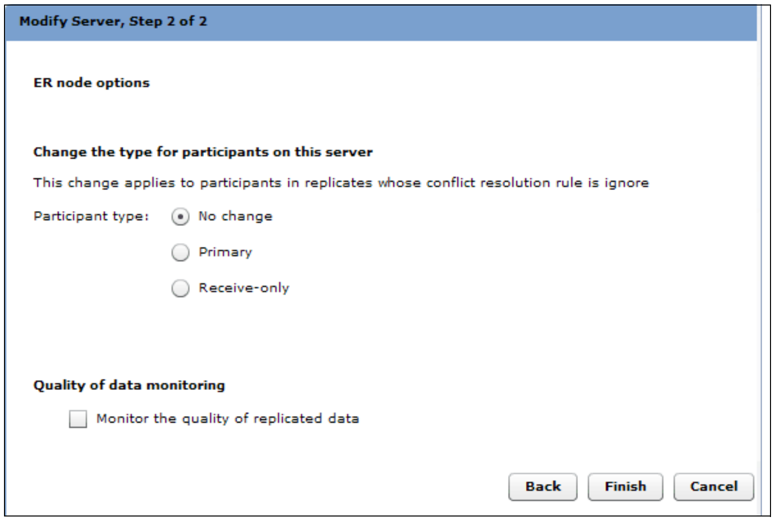
ATS directory: Disable ATS file generation [✓ Generate the ATS files in the temporary directory] Generate the ATS files in this directory:

RIS directory: Disable RIS file generation [✓ Generate the RIS files in the temporary directory] Generate the RIS files in this directory:

ATS and RIS file format: [✓ Text] XML Both

Idle timeout (minutes): [✓ Off] 0

Figure 5-30 OAT view of cdr modify server: Step 1


Modify Server, Step 2 of 2

ER node options

Change the type for participants on this server

This change applies to participants in replicates whose conflict resolution rule is ignore

Participant type: [✓ No change] Primary Receive-only

Quality of data monitoring

☐ Monitor the quality of replicated data

Figure 5-31 OAT view of cdr modify server: Step 2

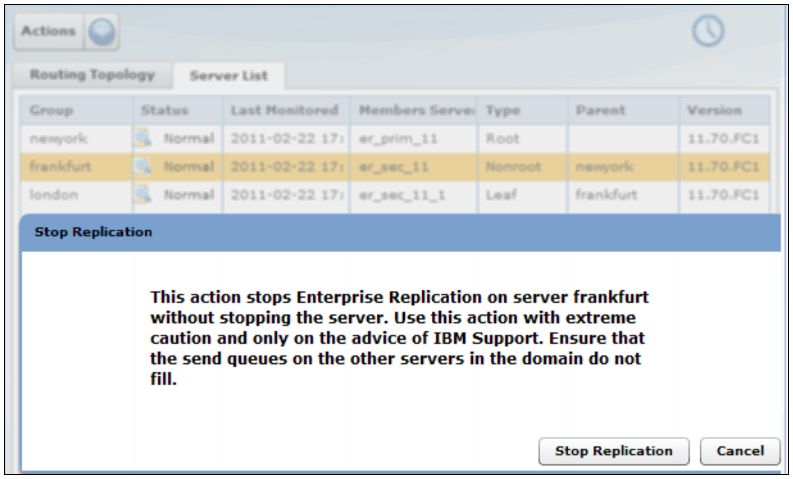
5.9.3 Starting and restarting the replication server

If there is a need to completely shut down the replication activities of a particular replication server in the ER domain, run the **`cdr stop`** command on that server.

The **`cdr stop`** command informs the replication server to stop capturing data to be replicated and also to stop receiving data from the other servers. So when this command is executed, all the database server threads related to ER are deleted.

Note: If ER is stopped in a root or a non-root node that has leaf nodes attached to it, then the flow of replication data to and from the leaf nodes are blocked.

To stop a replication server using OAT, select **Replication** → **ER Domain** → **frankfurt** → **Action** → **Stop Replication**. Figure 5-32 shows the OAT view of `cdr stop replication`.



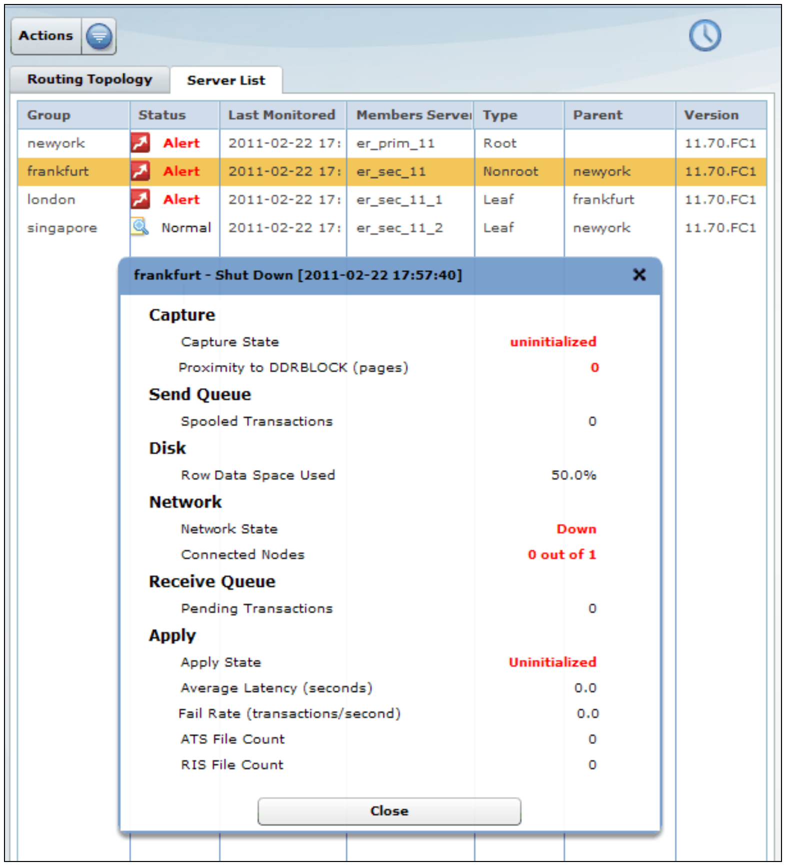
Group	Status	Last Monitored	Members Serve	Type	Parent	Version
newyork	Normal	2011-02-22 17:	er_prim_11	Root		11.70.FC1
frankfurt	Normal	2011-02-22 17:	er_sec_11	Nonroot	newyork	11.70.FC1
london	Normal	2011-02-22 17:	er_sec_11_1	Leaf	frankfurt	11.70.FC1

Stop Replication

This action stops Enterprise Replication on server frankfurt without stopping the server. Use this action with extreme caution and only on the advice of IBM Support. Ensure that the send queues on the other servers in the domain do not fill.

Figure 5-32 Stop replication

Figure 5-33 shows that the frankfurt database server has been shut down.



Group	Status	Last Monitored	Members Server	Type	Parent	Version
newyork	Alert	2011-02-22 17:	er_prim_11	Root		11.70.FC1
frankfurt	Alert	2011-02-22 17:	er_sec_11	Nonroot	newyork	11.70.FC1
london	Alert	2011-02-22 17:	er_sec_11_1	Leaf	frankfurt	11.70.FC1
singapore	Normal	2011-02-22 17:	er_sec_11_2	Leaf	newyork	11.70.FC1

frankfurt - Shut Down [2011-02-22 17:57:40]

Capture	
Capture State	uninitialized
Proximity to DDRBLOCK (pages)	0
Send Queue	
Spooled Transactions	0
Disk	
Row Data Space Used	50.0%
Network	
Network State	Down
Connected Nodes	0 out of 1
Receive Queue	
Pending Transactions	0
Apply	
Apply State	Uninitialized
Average Latency (seconds)	0.0
Fail Rate (transactions/second)	0.0
ATS File Count	0
RIS File Count	0

Figure 5-33 Server shut down using cdr stop

Similarly you can restart the replication by running **`cdr start`** command or in OAT by selecting **Replication** → **ER Domain** → **frankfurt** → **Action** → **Start Replication**.

5.9.4 Suspending and resuming a replication server

In this section, we demonstrate how to suspend and resume a replication server using the cdr utility and OAT.

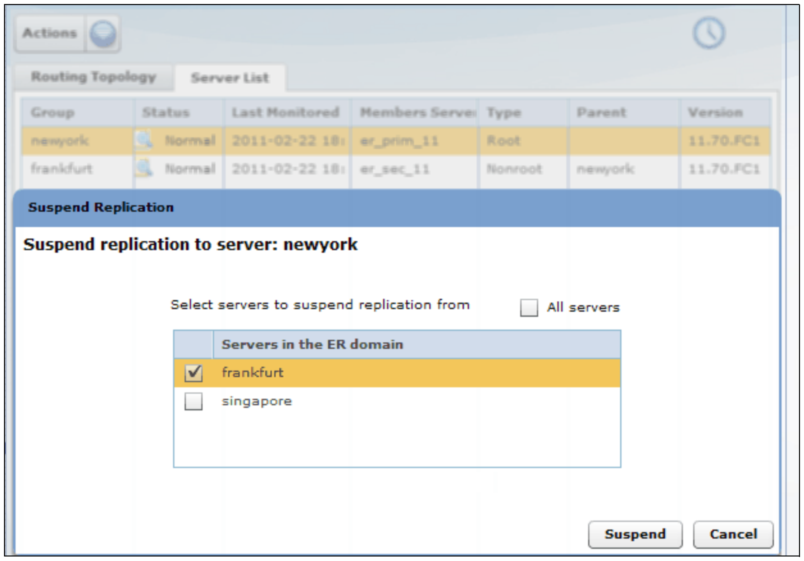
Suspending a replication server

If there is a need to stop the flow of replicating data at a particular server, the **`cdr suspend server <`*`to server`*`> <`*`from_server`*`>`** command is also an option. This command differs from the **`cdr stop`** command in many aspects. As examples, the **`cdr suspend server`** command:

- Stops the flow of replication data at a particular server, but the ER node still allows other control messages (such as control messages related to **`cdr define server`**, **`cdr define repl`**, and **`cdr start repl`**) and network acknowledgement messages to flow.
- Is more granular than the **`cdr stop`** command in the sense that you can either stop the flow of data to a particular server from other servers or stop the flow of data from the other servers to a particular server.
- If executed at a particular server, that server still continues to capture and put the replication data in its send queues so as to replicate to the other servers after it is resumed. But if the **`cdr stop`** command is issued at a particular server, then no capture is done at that server.

Important: When suspending a server, make sure that the send queues at the server are sufficiently sized so that you do not fill them up. For more information, refer to “Setting Up Send and Receive Queue Spool Areas” in the *IBM Informix Enterprise Replication Guide*, SC27-3539.

To suspend replication from OAT, select **Replication** → **ER Domain** → **newyork** → **Action** → **Suspend Replication.** Figure 5-34 shows the OAT view of cdr suspend replication.



Group	Status	Last Monitored	Members Serve	Type	Parent	Version
newyork	Normal	2011-02-22 18:	er_prim_11	Root		11.70.FC1
frankfurt	Normal	2011-02-22 18:	er_sec_11	Nonroot	newyork	11.70.FC1

Suspend Replication

Suspend replication to server: newyork

Select servers to suspend replication from ☐ All servers

	Servers in the ER domain
✓	frankfurt
☐	singapore

Figure 5-34 Suspend server

Resuming a replication server

To resume replication of data from a suspended server, use the **`cdr resume server`** ***`<to_server>`*** ***`<from_server>`*** command. To perform this action in OAT, select **Replication** → **ER Domain** → **newyork** → **Action** → **Resume Replication**.

5.9.5 Enabling and disabling a replication server

In this section, we show how to enable and disable a replication server using the cdr utility or OAT.

Enabling a replication server

You must synchronize the servers after you enable replication on it. Shutting down and restarting the disabled replication server does not enable replication.

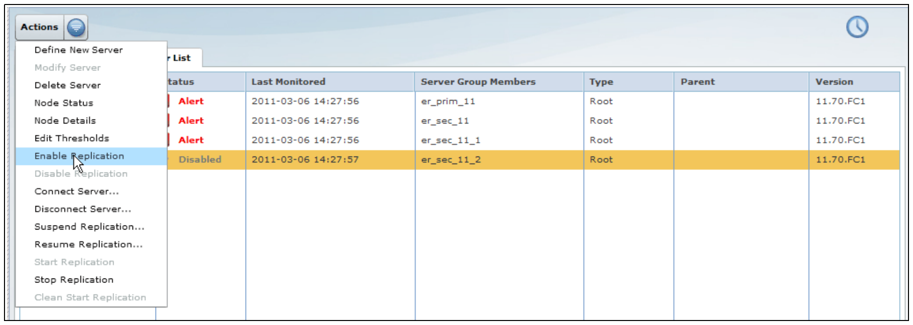
You can both enable and synchronize a disabled replication server by running the **`cdr check replicate set`** command with the --repair and the --enable options. Alternatively, you can run the **`cdr enable server`** command and then synchronize the server.

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 servers.

The following two commands enable the *singapore* replication server, which was disabled, and synchronize the server.

```
cdr enable server -c singapore singapore
cdr check replicateset --master=newyork --replset=myrepset singapore --repair
```

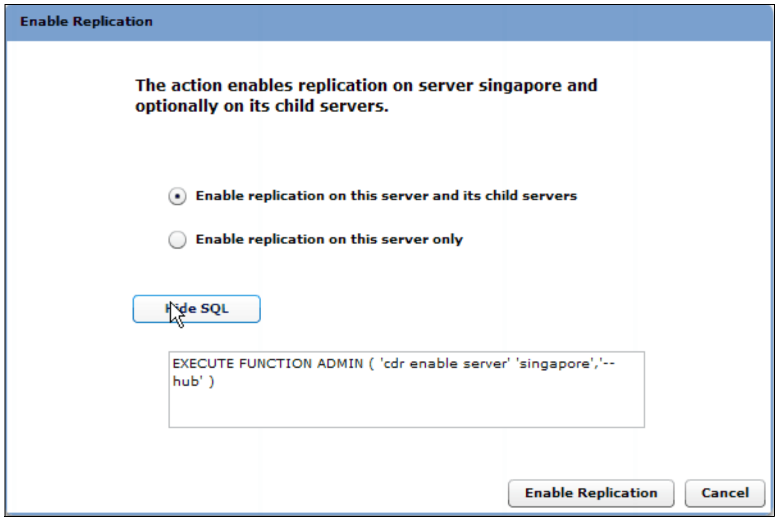
To perform this action in OAT, select **Replication** → **ER Domain** → **singapore** → **Action** → **Enable Server**, as shown in Figure 5-35.



tatus	Last Monitored	Server Group Members	Type	Parent	Version
Alert	2011-03-06 14:27:56	er_prim_11	Root		11.70.FC1
Alert	2011-03-06 14:27:56	er_sec_11	Root		11.70.FC1
Alert	2011-03-06 14:27:56	er_sec_11_1	Root		11.70.FC1
Disabled	2011-03-06 14:27:57	er_sec_11_2	Root		11.70.FC1

Figure 5-35 OAT view of cdr enable server: Step 1

In the next window, you can choose to enable both replication server and its child servers or only the replication server itself (Figure 5-36).



The action enables replication on server singapore and optionally on its child servers.

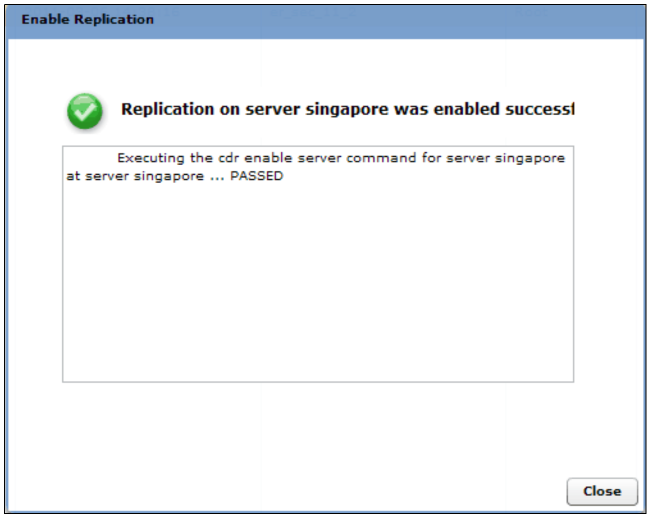
[✓ Enable replication on this server and its child servers]

Enable replication on this server only

```
EXECUTE FUNCTION ADMIN ( 'cdr enable server' 'singapore','--
hub' )
```

Figure 5-36 OAT view of cdr enable server: Step 2

OAT displays a confirmation window after the replication enabling is successful (Figure 5-37).

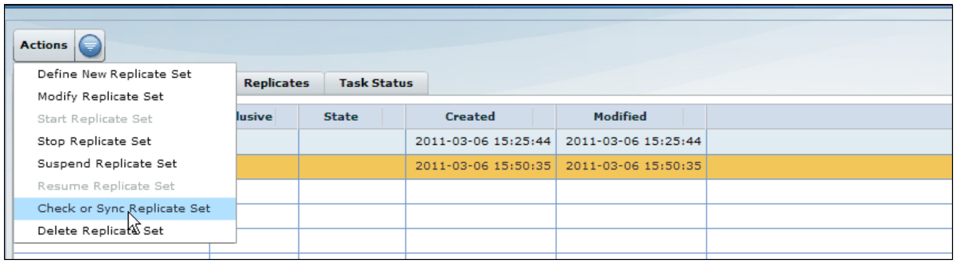


Replication on server singapore was enabled successl

```
        Executing the cdr enable server command for server singapore
at server singapore ... PASSED
```

Figure 5-37 OAT view of cdr enable server: Step 3

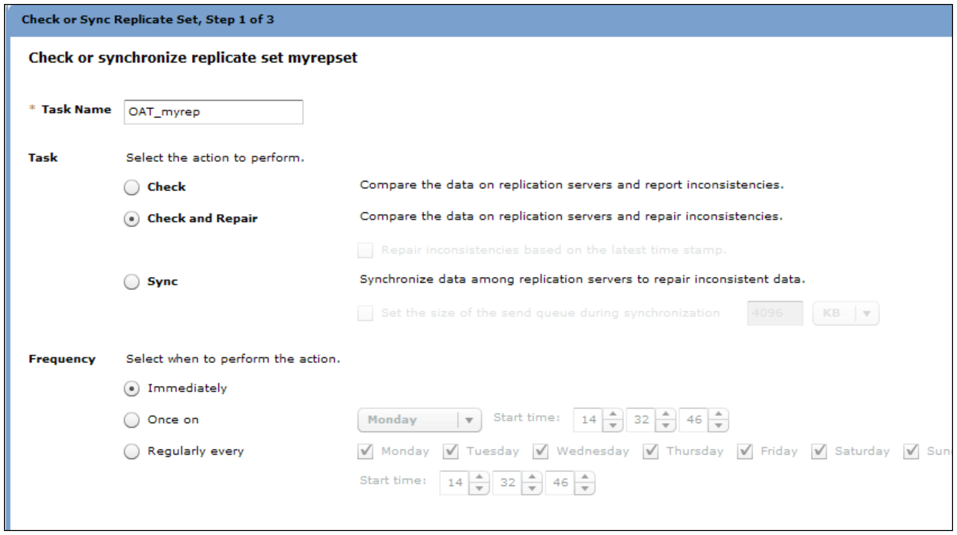
Now that the node is enabled, we need to synchronize the server. In OAT, select **Replicates** → **Replicate Set** → **Action** → **Check or Sync Replicate Set** (Figure 5-38).



lusive	State	Created	Modified
		2011-03-06 15:25:44	2011-03-06 15:25:44
		2011-03-06 15:50:35	2011-03-06 15:50:35

Figure 5-38 Check replicate to synchronize the data

In the Check or Sync Replication Set, Step 1 of 3 window (Figure 5-39), select **Check and Repair**.



Check or Sync Replicate Set, Step 1 of 3

Check or synchronize replicate set myrepset

* Task Name: OAT_myrep

Task — Select the action to perform.

- ○ Check — Compare the data on replication servers and report inconsistencies.
- ● Check and Repair — Compare the data on replication servers and repair inconsistencies.
  - ☐ Repair inconsistencies based on the latest time stamp.
- ○ Sync — Synchronize data among replication servers to repair inconsistent data.
  - ☐ Set the size of the send queue during synchronization 4096 KB

Frequency — Select when to perform the action.

- ● Immediately
- ○ Once on — Monday, Start time: 14 32 46
- ○ Regularly every — ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sun
  - Start time: 14 32 46

Figure 5-39 cdr check replicate to synchronize the data: Step 1

In the Check or Sync Replication Set, Step 2 of 3 window (Figure 5-40), specify the limits.

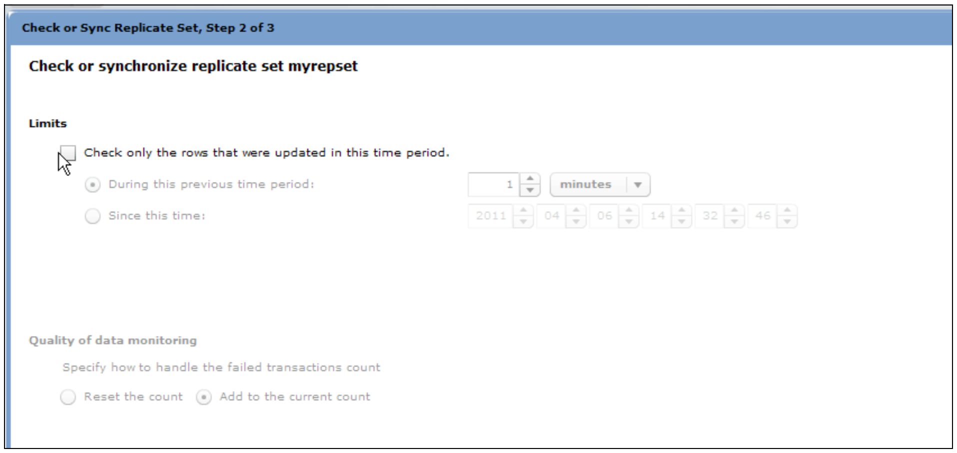
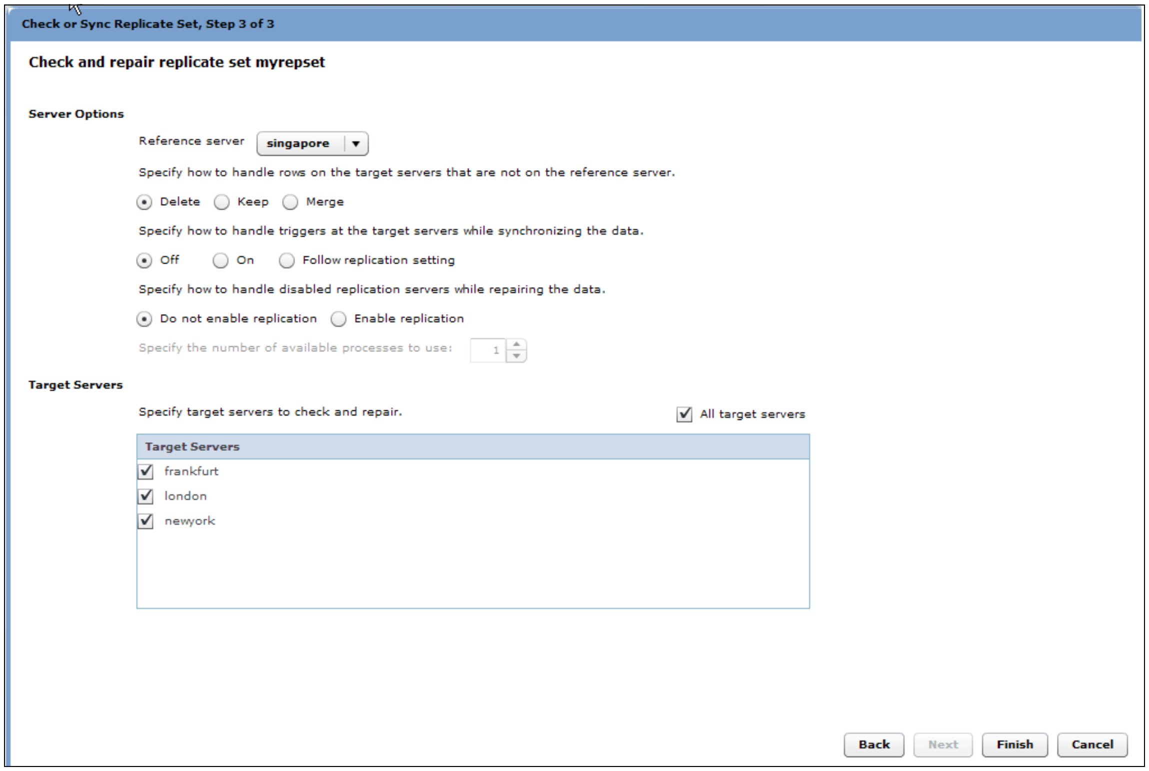

Check or Sync Replicate Set, Step 2 of 3
Check or synchronize replicate set myrepset
Limits
Check only the rows that were updated in this time period.
During this previous time period:
1
minutes
Since this time:
2011
04
06
14
32
46
Quality of data monitoring
Specify how to handle the failed transactions count
Reset the count
Add to the current count

Figure 5-40 cdr check replicate to synchronize the data: Step 2

In the Check or Sync Replication Set, Step 3 of 3 window (Figure 5-41), specify server options and select target servers.


Check or Sync Replicate Set, Step 3 of 3

Check and repair replicate set myrepset

Server Options

Reference server: singapore

Specify how to handle rows on the target servers that are not on the reference server.

[✓ Delete] Keep Merge

Specify how to handle triggers at the target servers while synchronizing the data.

[✓ Off] On Follow replication setting

Specify how to handle disabled replication servers while repairing the data.

[✓ Do not enable replication] Enable replication

Specify the number of available processes to use: 1

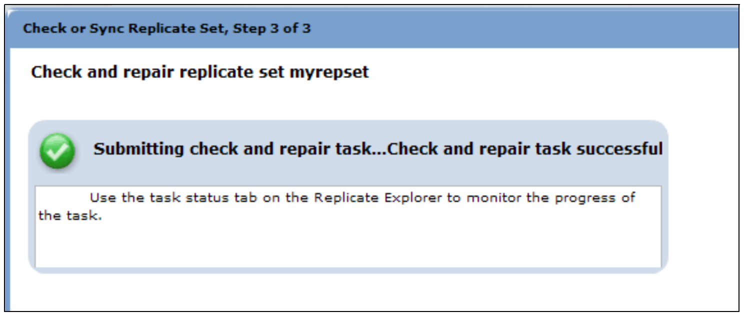
Target Servers

Specify target servers to check and repair. ✓ All target servers

Target Servers
✓ frankfurt
✓ london
✓ newyork

Figure 5-41 cdr check replicate to synchronize the data: Step 3

OAT displays a successful message after the synchronization is complete (Figure 5-42).


Check or Sync Replicate Set, Step 3 of 3

Check and repair replicate set myrepset

Submitting check and repair task...Check and repair task successful

Use the task status tab on the Replicate Explorer to monitor the progress of the task.

Figure 5-42 cdr check replicate to synchronize the data complete

Disabling a replication server

If you need to temporarily stop replication and your replicates, use the time stamp or delete wins conflict resolution rule and then use the **`cdr disable server`** command.

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.

Disabling replication has the following effects:

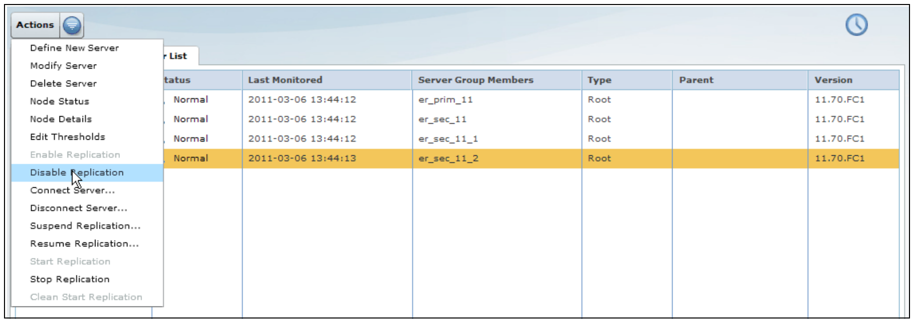
- There is no connection between the disabled replication server and active replication servers.
- Transactions on the disabled replication server are not queued for replication.
- Transactions on active replication servers are not queued for the disabled replication server.
- Control messages on active 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

The following command disables the server, singapore, which is connected to the replication domain:

```
cdr disable server -c singapore singapore
```

Note: 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.

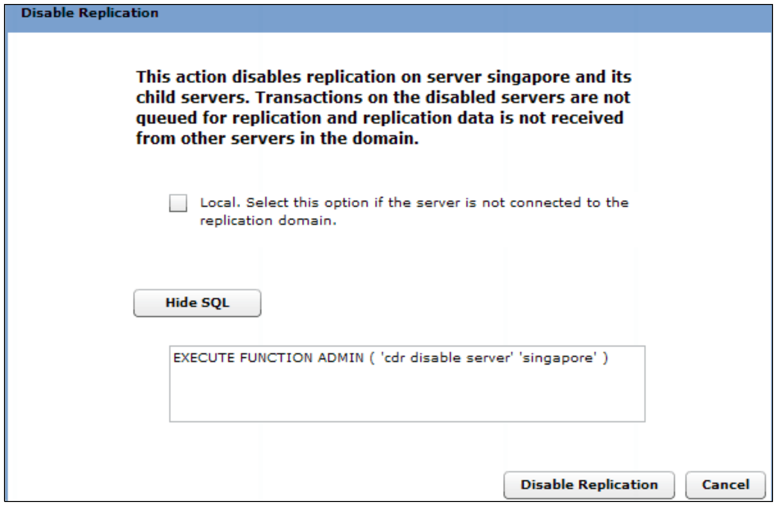
To disable the singapore server using OAT, select **Replication** → **ER Domain** → **singapore** → **Action** → **Disable Server** (Figure 5-43).



tatus	Last Monitored	Server Group Members	Type	Parent	Version
Normal	2011-03-06 13:44:12	er_prim_11	Root		11.70.FC1
Normal	2011-03-06 13:44:12	er_sec_11	Root		11.70.FC1
Normal	2011-03-06 13:44:12	er_sec_11_1	Root		11.70.FC1
Normal	2011-03-06 13:44:13	er_sec_11_2	Root		11.70.FC1

Figure 5-43 OAT view of cdr disable server: Step 1

In the first Disable Replication window (Figure 5-44), select **Disable Replication**.



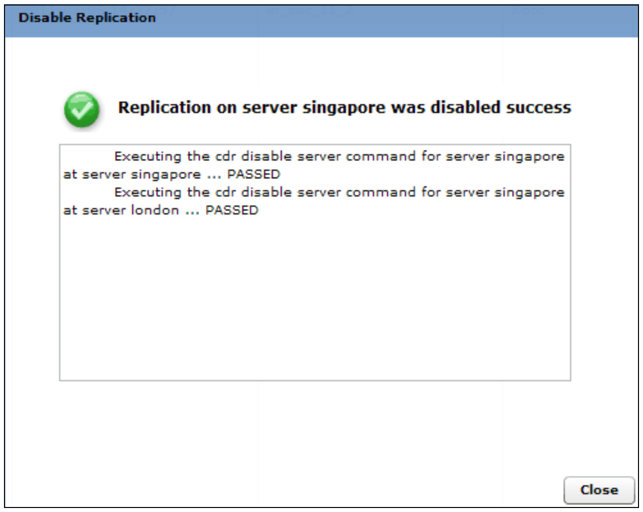
This action disables replication on server singapore and its child servers. Transactions on the disabled servers are not queued for replication and replication data is not received from other servers in the domain.

☐ Local. Select this option if the server is not connected to the replication domain.

```
EXECUTE FUNCTION ADMIN ( 'cdr disable server' 'singapore' )
```

Figure 5-44 OAT view cdr disable server: Step 2

OAT displays a successful message after the server is disabled (Figure 5-45).

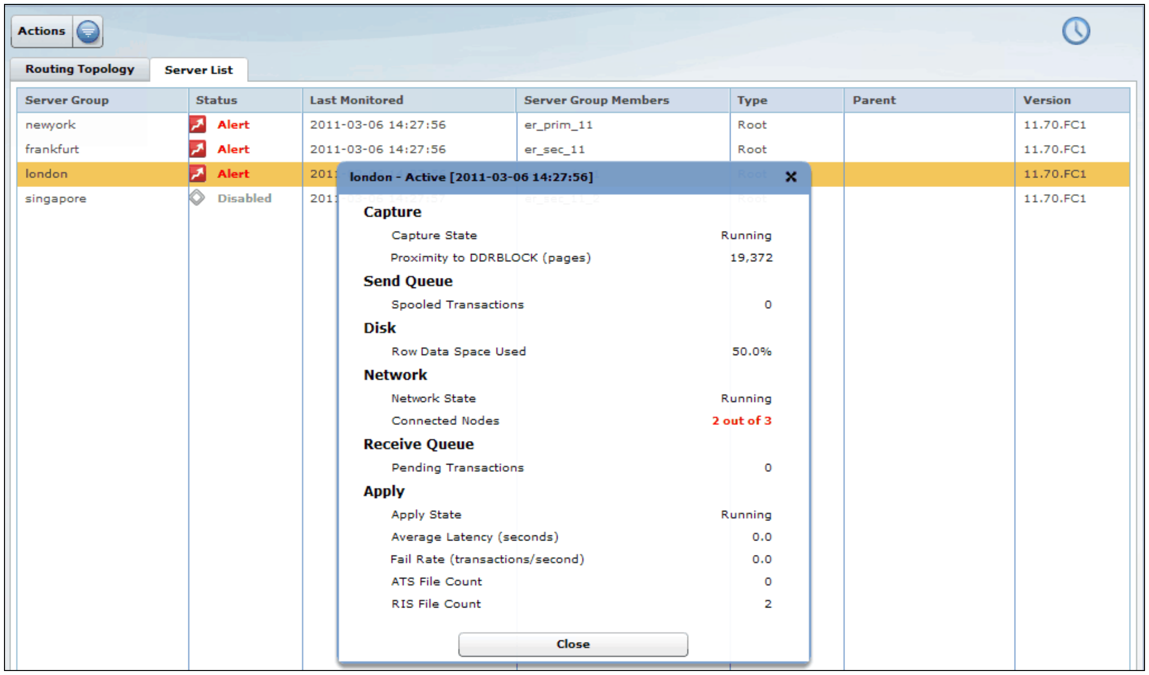


Replication on server singapore was disabled success

```
        Executing the cdr disable server command for server singapore
at server singapore ... PASSED
        Executing the cdr disable server command for server singapore
at server london ... PASSED
```

Figure 5-45 OAT view of cdr disable server: Step 3

From OAT, you can view the detailed status of the servers in the cluster after one sever is disabled (Figure 5-46).



Server Group	Status	Last Monitored	Server Group Members	Type	Parent	Version
newyork	Alert	2011-03-06 14:27:56	er_prim_11	Root		11.70.FC1
frankfurt	Alert	2011-03-06 14:27:56	er_sec_11	Root		11.70.FC1
london	Alert	201				11.70.FC1
singapore	Disabled	201				11.70.FC1

london - Active [2011-03-06 14:27:56]

Capture	
Capture State	Running
Proximity to DDRBLOCK (pages)	19,372
Send Queue	
Spooled Transactions	0
Disk	
Row Data Space Used	50.0%
Network	
Network State	Running
Connected Nodes	2 out of 3
Receive Queue	
Pending Transactions	0
Apply	
Apply State	Running
Average Latency (seconds)	0.0
Fail Rate (transactions/second)	0.0
ATS File Count	0
RIS File Count	2

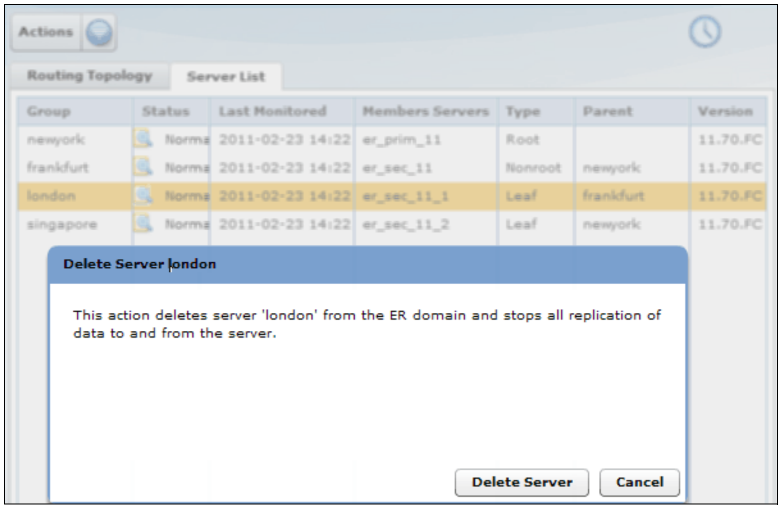
Figure 5-46 OAT view showing the server is disabled

5.9.6 Deleting a replication server

Even after careful topology planning, there will be times when a replication server is no longer needed. At that time, the **`cdr delete server`** command can be used to delete the server.

Note: Due to the destructive and un-repairable nature of deleting a server, the **`cdr delete server`** command must be run twice: once on the local server and once on the remote server. Prior to running this command, review your topology to ensure that the delete will not affect any other nodes. Also, do not delete a replication server and then immediately recreate a replication server with the same name.

To delete a replication server in OAT, select **Replication** → **ER Domain** → **London** → **Action** → **Delete Server**. Figure 5-47 shows the OAT view of the **`cdr delete server`** command.



Group	Status	Last Monitored	Members Servers	Type	Parent	Version
newyork	Norma	2011-02-23 14:22	er_prim_11	Root		11.70.FC
frankfurt	Norma	2011-02-23 14:22	er_sec_11	Nonroot	newyork	11.70.FC
london	Norma	2011-02-23 14:22	er_sec_11_1	Leaf	frankfurt	11.70.FC
singapore	Norma	2011-02-23 14:22	er_sec_11_2	Leaf	newyork	11.70.FC

Delete Server london

This action deletes server 'london' from the ER domain and stops all replication of data to and from the server.

Figure 5-47 cdr delete server

5.10 Maintaining replicates

Many times the replicates need fine tuning, or a business request comes in that requires a replicate to be modified. A replicate contains attributes or participants, properties, and states, and all these objects can be modified.

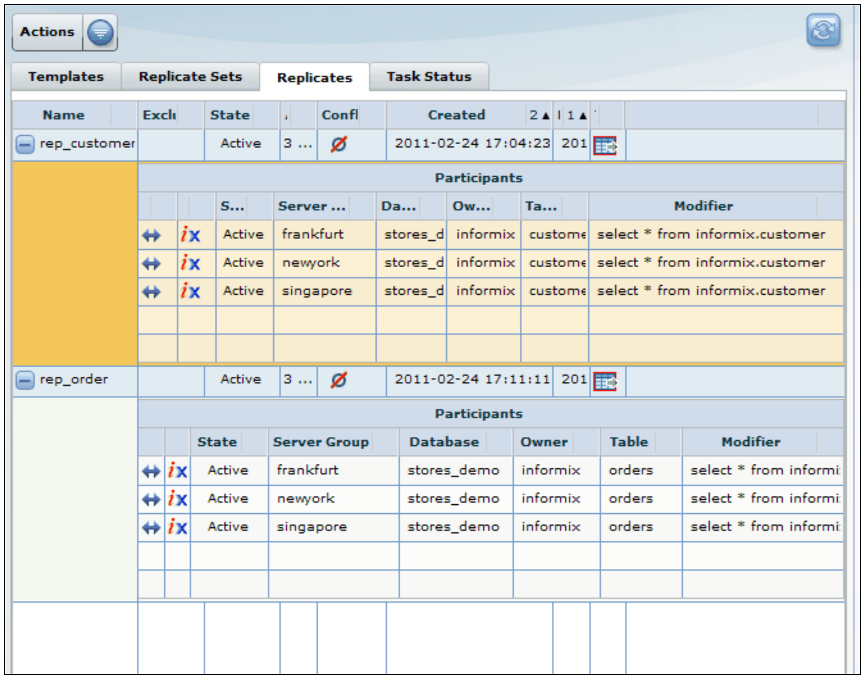
In this section, we demonstrate how to modify a replicate that has all the columns (SELECT *) of a table in the replicate to just select few columns from the table.

To do this task, we first have to discover the name of the replicate using the **`cdr list replicate`** command, followed by the **`cdr change replicate`** command.

5.10.1 Listing replicates

To find all the replicates defined, run the **`cdr list replicate`** command. By executing this command, and not specifying the name of a replicate, all replicates defined on the server are listed. Use the **`cdr list replicate`** ***`<replicate_name>`*** command if the name of the replicate is known.

To list the replicates from OAT, select **Replicates(L)** → **Replicate**. Figure 5-48 shows the OAT view of **`cdr replicate`**.



Name	Excl	State		Confl	Created	2 ▲	1 ▲
⊟ rep_customer		Active	3 ...	Ø	2011-02-24 17:04:23	201	▦

Participants

		S...	Server ...	Da...	Ow...	Ta...	Modifier
↔	ix	Active	frankfurt	stores_d	informix	custome	select * from informix.customer
↔	ix	Active	newyork	stores_d	informix	custome	select * from informix.customer
↔	ix	Active	singapore	stores_d	informix	custome	select * from informix.customer

Name	Excl	State		Confl	Created	2 ▲	1 ▲
⊟ rep_order		Active	3 ...	Ø	2011-02-24 17:11:11	201	▦

Participants

		State	Server Group	Database	Owner	Table	Modifier
↔	ix	Active	frankfurt	stores_demo	informix	orders	select * from informi
↔	ix	Active	newyork	stores_demo	informix	orders	select * from informi
↔	ix	Active	singapore	stores_demo	informix	orders	select * from informi

Figure 5-48 cdr list replicate

5.10.2 Modifying replicates

In this section, we show how to change a replicate to select only a few columns of a table instead of all the columns (SELECT *) from a table.

In Example 5-48 on page 198, the **`cdr change replicate`** command is used. This command should not be confused with the other possible command, **`cdr modify replicate`**, which makes changes only to the replicate attributes, such as conflict options, scope options, frequency options, and mode.

This **`cdr change replicate`** command enables you to change the SELECT statement. However, the only way to do so is to drop and add the participant using the correct SELECT statement.

Example 5-48 shows how the **cdr change replicate** command is used to modify the replicates.

Example 5-48 cdr change replicate

```
xmach6:/usr/informix>cdr list replicate brief rep_order

REPLICATE           TABLE                                    SELECT
---------------------------------------------------------------------------
rep_order           stores_demo@newyork:informix.orders      select * from informix.orders
rep_order           stores_demo@singapore:informix.orders    select * from informix.orders
rep_order           stores_demo@frankfurt:informix.orders    select * from informix.orders

xmach6:/usr/informix>cdr change replicate -d rep_order "stores_demo@newyork:informix.orders"
"stores_demo@singapore:informix.orders"
"stores_demo@frankfurt:informix.orders"

xmach6:/usr/informix>cdr list replicate brief rep_order

REPLICATE           TABLE                                    SELECT
---------------------------------------------------------------------------
rep_order


xmach6:/usr/informix>cdr change replicate -a rep_order "stores_demo@newyork:informix.orders" "select
order_num, customer_num from orders"
"stores_demo@singapore:informix.orders" "select order_num, customer_num from orders"
"stores_demo@frankfurt:informix.orders" "select order_num, customer_num from orders"

xmach6:/usr/informix>cdr list replicate brief rep_order

REPLICATE       TABLE                                      SELECT
-------------   ---------------------------------------   -----------------------------------------
rep_order       stores_demo@newyork:informix.orders        select order_num, customer_num from orders
rep_order       stores_demo@singapore:informix.orders      select order_num, customer_num from orders
rep_order       stores_demo@frankfurt:informix.orders      select order_num, customer_num from orders
```

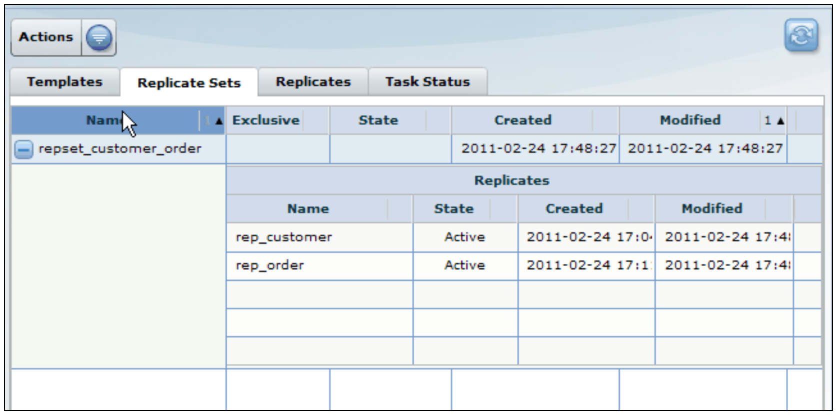
5.11 Maintaining replicate sets

Replicate sets allow you to group many replicates together for easier management. By grouping these replicates together into a replicate set, you are able to list, start, stop, suspend, and resume all the replicates within a replicate set at the same time.

5.11.1 Listing replicate sets

To discover which replicate sets exist and which replicates are included within each set, use the **cdr list replicateset** command.

To list replicate sets from OAT, select **Replicates(L)** → **Replicate Sets (R)**. Figure 5-49 shows the OAT view of **`cdr list replicateset`**. In this figure, the set_customer_order replicate set is listed, which consists of the rep_customer and rep_order replicate.



Name	Exclusive	State	Created	Modified
repset_customer_order			2011-02-24 17:48:27	2011-02-24 17:48:27

Replicates

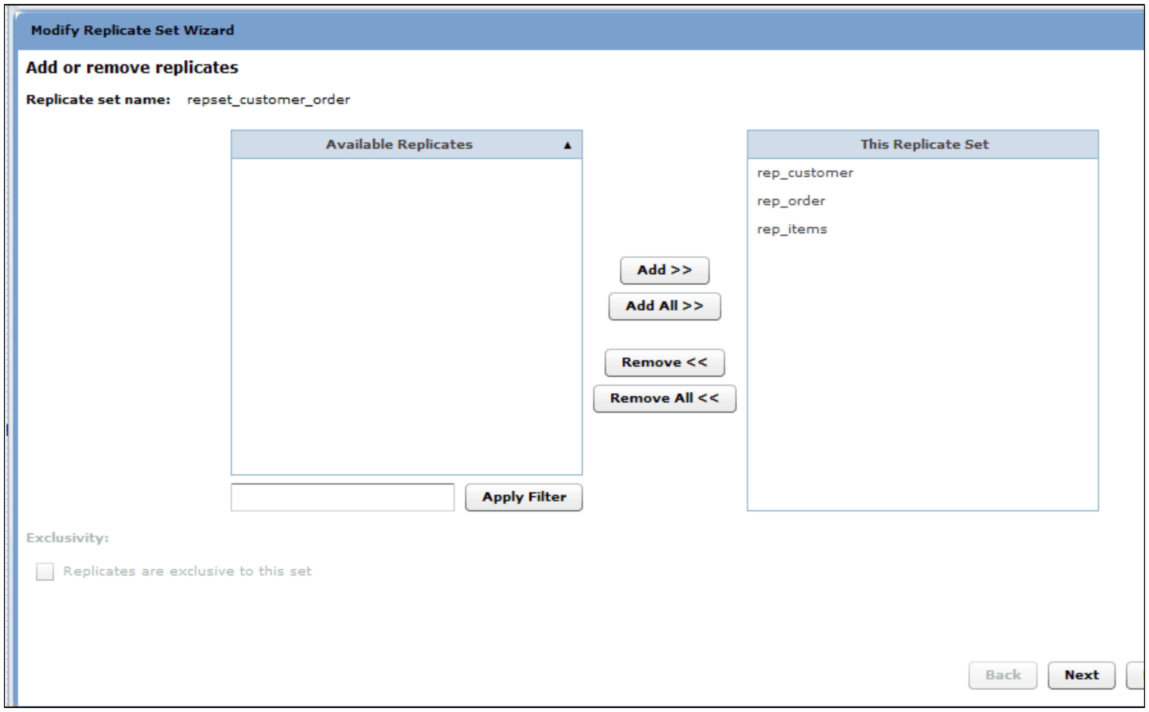
Name	State	Created	Modified
rep_customer	Active	2011-02-24 17:0	2011-02-24 17:4
rep_order	Active	2011-02-24 17:1	2011-02-24 17:4

Figure 5-49 cdr list replicateset

5.11.2 Modifying replicate sets

Replicate sets can only be modified by adding and deleting replicates within the replicate set or changing the replication frequency. Remember that when modifying any replicate set that every replicate in the set will be affected. The **`cdr change replicateset`** command can be used to change the replicate set.

In Figure 5-49 on page 199, the repset_customer_order replicate set was defined using the rep_customer and rep_order replicates. Due to business needs, there is a requirement to update the items table. To perform this action in OAT, select **Replicate(L)** → **Replicate Set** → **select replicateset** → **Action** → **Modify Replicate Set?.** Figure 5-50 shows that repset_customer_order is changed to accommodate the new requirement.


Modify Replicate Set Wizard

Add or remove replicates

Replicate set name: repset_customer_order

Available Replicates

This Replicate Set
rep_customer
rep_order
rep_items

Exclusivity:

Replicates are exclusive to this set

Figure 5-50 cdr change replicateset

5.11.3 Considerations for stopping or suspending a replicate set

In this section, we discuss some of the considerations for stopping or suspending a replicate set. After a replicate set has been started, avoid stopping or suspending it, which basically stops and suspends all the replicates. Even if the replicate set is created as an exclusive set, stopping or suspending a replicate set should be avoided. In a situation where time-based replication is being implemented, stopping and suspending a replicate set or replicate becomes even more dangerous.

When a replicate or replicate set is stopped or suspended, a situation called a *split transaction* can occur. When a stop or suspend command is issued, ER stops sending information, even if it is in the middle of a transaction.

For example, if table M and table C are being updated within a transaction, table M could be updated but table C might not be updated. ER breaks the transaction into separate pieces, but it cannot guarantee that after replication has resumed, all the pieces will be sent in the correct order. Therefore, as an example, children rows could arrive before the parent row. If referential integrity is turned on, then an error will be received and the child will be rolled back. In addition, the parent part of the transaction could come later and it could be successfully replicated, thus creating incorrect data. The referential integrity constraints are the primary reason why stopping or suspending a replicate or replicate set should be avoided.

The best practice when using ER is to let ER take care of itself, along with careful planning and monitoring. The planning phase is to make sure there is enough queue space set aside for the specific situation where a replication server is down.

Determining how much data is replicated per hour and how many hours are going to be allowed to queue determines how much queue space is needed. After replication resumes, re-syncing a server that was down is quite fast. We have seen cases with a financial company where a site was down for three to five hours, with thousands of transactions queuing up. After the site was back up, re-syncing the data only took about 30 to 45 minutes.

In cases where the amount of data being queued up is beyond the desired business requirement, you can delete the replication server altogether using the **`cdr delete server`** command, and then resync using the **`cdr sync`** or **`cdr check sync`** commands.

Another solution to address a replication server that is down is to create replicates and replicate sets using templates. When a server is deleted, there can be many replicates that need to be recreated. If templates are not used, this can be a tedious task. By using templates, all the replicates and replicate sets are created and just need to be realized by using the **`cdr realize template`** command and all the replicates that have been deleted will be recreated again.

5.12 Maintaining templates

Templates make administering Enterprise Replication much easier. They provide a simple way to set up and deploy a large set of tables for Enterprise Replication. After a template is created, it sets up replicates, performs initial data synchronization, and even creates tables during realization if the tables do not exist on the target servers.

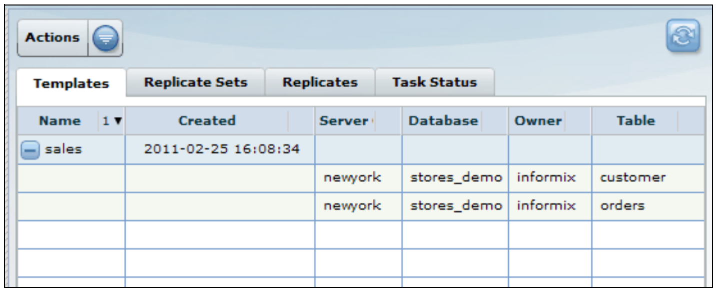
Maintaining templates is about as simple as creating them. There are two commands available:

- **`cdr list template`**
- **`cdr delete template`**

5.12.1 Listing a template

After a template is created, to see the components of the template, use the **`cdr list template`** command. This command lists all the templates created. If more details about a particular template is needed, simply add the name of the template at the end of the command.

To list templates with OAT, select **Replicate(L)** → **Template**. Figure 5-51 shows the OAT view of **`cdr list template`**.



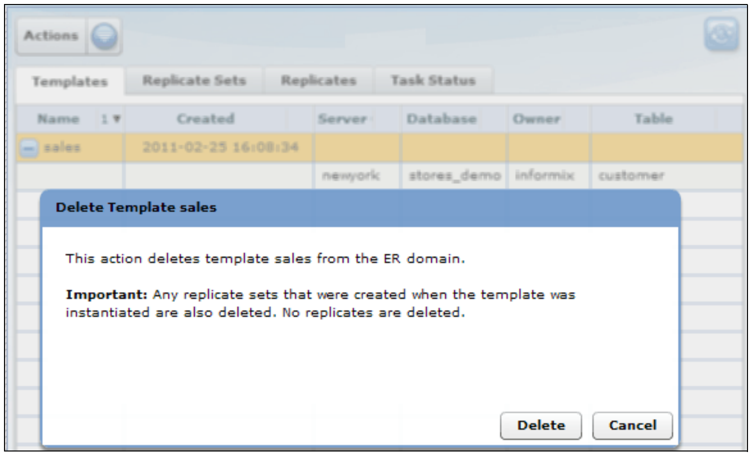
Name 1 ▼	Created	Server	Database	Owner	Table
sales	2011-02-25 16:08:34				
		newyork	stores_demo	informix	customer
		newyork	stores_demo	informix	orders

Figure 5-51 cdr list template

5.12.2 Deleting a template

If a template requires changes, the only way to do so is to first delete the template using the **`cdr delete template`** command and then re-create it.

To delete a template in OAT, select **Replicate(L)** → **Template** → **Select Template** → **Action** → **Delete Template**. Figure 5-52 shows an OAT view of **`cdr delete template`**.



Name	Created	Server	Database	Owner	Table
sales	2011-02-25 16:08:34				
		newyork	stores_demo	informix	customer

Delete Template sales

This action deletes template sales from the ER domain.

Important: Any replicate sets that were created when the template was instantiated are also deleted. No replicates are deleted.

Figure 5-52 cdr delete template

Important: If a template is deleted after it has been created but before it has been realized, then all the replicates created as part of that template creation are also deleted. If a template is deleted after it has been realized, then even if the template is deleted, the replicates are retained by ER and replication using those replicates continue. You have to delete each of the replicates using the **`cdr delete replicate`** command if you want to remove all the replicates created by the define or realize template commands.

5.13 Altering tables with defined replicates

Performing the ALTER operations on a replicated table does not require additional steps or care, but requires at least the replicate defined as a mastered replicate. For more information about these requirements, see “Managing Replication Servers and Replicates” of the *IBM Informix Database Server Guide* at the following address:

`http://publib.boulder.ibm.com/infocenter/idshelp/v117/index.jsp`

Except for fragment attach, all other ALTER operations do not require a user to explicitly put the table in the cdr alter mode using the **`cdr alter`** command.

Here we demonstrate how to add a column to a replicated table. We add a new column myco to the customer table that is replicated in three database servers.

To add a column to a replicated table, complete the following steps:

1. Obtain the replicate name using the **cdr list repl** command.

   Example 5-49 shows how to obtain the replicate name on the customer table.

   Example 5-49 Obtain the replicate name

```
xmach6:/gpfs1/informix>cdr list replicate |more
CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        repmaster_customer
STATE:            Active ON:newyork
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores_demo:informix.customer
OPTIONS:          transaction,ris,ats,floatieee,fullrow
REPLID:           192020500 / 0xb720014
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork
REPLTYPE:         Master
```

2. Get the list of ER nodes from which the table is used as part of the replicate.

   In Example 5-50, we list repmaster_customer and the table column shows the nodes involved.

   Example 5-50 List the ER nodes the table issued

```
xmach6:/gpfs1/informix>cdr list replicate brief repmaster_customer

REPLICATE           TABLE                                     SELECT
------------------- ---------------------------------------  ----------------------------------
repmaster_customer   stores_demo@newyork:informix.customer    select  customer_num, fname, lname,
                                                              company, address1, address2, city,
                                                              state, zipcode, phone from customer
repmaster_customer   stores_demo@frankfurt:informix.customer select  customer_num, fname, lname,
                                                              company, address1, address2, city,
                                                              state, zipcode, phone from customer
repmaster_customer   stores_demo@singapore:informix.customer  select  customer_num, fname, lname,
                                                               company, address1, address2, city,
                                                               state, zipcode phone from customer
```

3. Alter the table to add a column in each node.

   The **`alter table`** command shown in Example 5-51 should be run at the newyork, frankfurt, and singapore nodes.

   Example 5-51 Altering a table to add a column

```
xmach6:/gpfs1/informix>dbaccess stores_demo -
Database selected.
> alter table customer add mycol integer;
Table altered.
```

   At this point, replication on the customer table continues, but the value populated in the new column at one node is not propagated to other nodes. In Example 5-52, we populated the mycol value as 100 in the region1 node and selected the row at the HQ node. You can see that the row is replicated from region1 to HQ but the column value mycol is not replicated at HQ.

   Example 5-52 Insert a record and check if the data is replicated

```
xmach6:/gpfs1/informix>dbaccess stores9@region1 -
Database selected.
> insert into customer values
(0,"NIMMU",NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,100);
1 row(s) inserted.

xmach6:/gpfs1/informix>dbaccess stores9@HQ -
Database selected.
> select * from customer where fname="NIMMU";
customer_num 131
fname NIMMU
lname
company
address1
address2
city
state
zipcode
phone
mycol
1 row(s) retrieved.
>
Note that the mycol value at HQ was populated with the DEFAULT value.
```

4. Remaster the replicate to include the new column in the select clause (Example 5-53).

Example 5-53 Remaster the replicate to include a new column

```
xmach6:/gpfs1/informix>cdr remaster -M newyork repmaster_customer “select *
from customer”
After this if you update the value in mycol at any of the nodes, the
customer table at all other nodes will see the new values.
```

Remastering is the process where the ER can check and update the dictionary information of a replicated table, especially after an ALTER operation on that table. Example 5-53 shows how this process works. After altering the table by adding a new column to customer table, the **`cdr remaster`** command is used to update the information of the existing replicate on that table.

A non-mastered replicate (classic replicate) can be converted into a mastered replicate using the **`cdr remaster`** command.

If a mastered replicate was defined with different column names of the participant tables, use the **`cdr swap shadow`** command to do manual remastering after the ALTER table operation.

A detailed example of the usage of the **`cdr swap shadow`** command can be found in the IBM developerWorks article “Schema evolution in an Enterprise Replication environment”, which can be found at the following address:

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0506inturi/index.html

Note: The mastered replicate creation process of **`cdr define repl`** checks the column names between the participants along with other dictionary information. You can use the --name=y|n option to disable the column name comparison. This action is useful in a scenario where the replicate participant tables have the same column types but different column names, but where you still want to make use of master replicate characteristics.

5.14 Altering replicated tables

The Flexible Grid has made administration of the replication servers much easier. Using Flexible Grid, a DBA can manage all the replication servers from a single point. Database modification, such as table schema changes, made in the grid context are global in nature. That means change is propagated to all the database servers participating in the grid. Use the Flexible Grid if you want to deploy schema changes instantaneously on all the database servers involved in replication.

However, you might need to perform schema changes in a non-grid environment, or in a phased manner because you have to deploy new applications along with the schema changes. In this section, we show you how to perform schema changes in a non-grid environment as well as things to keep in mind while making schema changes. Refer to 13.1.2, “Upgrading database schema and applications” on page 504 for information about how to deploy schema changes and application in a phased manner.

5.14.1 Alter table example

You can modify the replicated table schema using a standard ALTER SQL statement. No other step is necessary if the change is local and does not affect the replication attributes. Some examples are modifying default values, next extent size, table lock granularity, and add or drop constraints.

If the schema change is such that it affects the replication, such as add or drop columns, or modifying the data type of a replicated column, you need to update the replicate definition to reflect the schema change in the replication environment.

Typically, the procedure to alter replicated table is:

1. Alter the table using a standard SQL statement.
2. Update the replicate definition to reflect the change.

Here we demonstrate how to add a new column named mynewcol to the customer table:

1. Obtain the replicate name and associated table using the **`cdr list repl brief`** command. Example 5-54 shows the replicate name repmaster_customer and associated table customer. Table customer is replicated among three replication servers, that is, newyork, frankfurt, and singapore.

Example 5-54 Replicate name and its participants

```
$ cdr list replicate brief repmaster_customer

REPLICATE           TABLE                                     SELECT
------------------- --------------------------------------   ---------------------------------
repmaster_customer   stores_demo@newyork:informix.customer   select  customer_num, fname, lname,
                                                             company, address1, address2, city,
                                                             state, zipcode, phone from customer
repmaster_customer   stores_demo@frankfurt:informix.customer select  customer_num, fname, lname,
                                                             company, address1, address2, city,
                                                             state, zipcode, phone from customer
repmaster_customer   stores_demo@singapore:informix.customer  select  customer_num, fname, lname,
                                                              company, address1, address2, city,
                                                              state, zipcode phone from customer
```

2. Alter the table to add the new column. Example 5-51 on page 205 shows the ALTER TABLE command to add a column.

Example 5-55 Alter a table to add a column

```
$ dbaccess stores_demo -
Database selected.
> alter table customer add mynewcol integer;
Table altered.
```

At this point, replication on the customer table continues, but data changes to the newly added column are not replicated. To replicate the data changes from the new column, the table must be altered on all the database servers and replicate definition must be updated to include newly added for replication.

3. The process of updating the replicate definition is know as remastering. To update the replicate definition, use the **`cdr remaster`** command. Remaster the replicate to include the new column in the select clause. Example 5-53 on page 206 shows the **`remaster`** command.

Example 5-56 Remaster the replicate to include a new column

```
$ cdr remaster -M newyork repmaster_customer "select * from customer"
```

After the replicate definition is updated (remaster), data changes to the new columns are replicated to all the database servers.

5.14.2 Other considerations

When you work on the replicates, consider the following actions:

- Drop a replicated column.

  To drop a replicated column, the replicate definition must be updated first before you perform the actual drop column operation. Update the replicate definition to remove the column from the SELECT clause of the replicate definition, which stops replicating data changes for that column. Then you can drop the actual column using the ALTER TABLE statement on all the servers.

- Truncate the replicated table.

  In the case of TRUNCATE TABLE, the bulk delete operation is not replicated to the other replication server.

- Drop a replicated table.

  You cannot drop the replicated table while replication is active for that table. To drop the replicated table, first stop the replication for that table, then drop the replicate followed by the actual drop table action.

- Rename a replicated table.

  By default, master replicates have the name verification attribute turned on. This attribute allows ER to maintain the same column and table name across all the replication servers where a table is being replicated. But to rename a column or a table, the name verification attribute must be disabled. Otherwise, the rename operation will fail with error `26023: Cannot perform rename operation while non-mastered or strict mastered replicate is defined`. Example 5-57 shows how to disable the name verification attribute.

  Example 5-57 Disable name verification attribute

  ```
  $ cdr modify repl --name=n repmaster_customer
  ```

  After the name verification is disabled (turned off), then we can rename a column or a table using a RENAME statement.

- Rename a replicated database.

  A name verification attribute is not applicable to a database name. Thus, a database name can be changed regardless of a name verification attribute change.

- Manual remastering.

  If a mastered replicate was defined with the column names of participant tables being different, use the **`cdr swap shadow`** command to do manual remastering after the ALTER table operation.

For additional details about online schema changes, refer to the IBM developerWorks article “Schema evolution in an Enterprise Replication environment”, which can be found at the following address:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0506inturi/index.html`

In general, we would advise you to use grid functionality to perform the schema changes. It significantly reduces the efforts and time needed to complete the schema changes globally across all the database servers in a replication environment. For more details about administration in grid, refer to 4.3, “Administration tasks” on page 91.

5.15 Data synchronization options

There are times where you have to override the normal replication process. For example, the first time a replication is defined on a table between two servers, an initial synchronization between the servers should be performed. Otherwise, if the initial copies of the data are inconsistent, they remain inconsistent.

There are other reasons that you might want to perform a synchronization (or even a resynchronization) of the data:

- One of the servers has been out of service for a period of time. Rather than let data queue up for that server, you can chose to remove it from replication. As part of re-adding that server, you then resynchronize the data.
- Some rows failed to be applied on a server. This situation could have occurred because of some type of conflict, or possibly some environmental issue such as running out of disk space. After resolving the problem, you should repair the inconsistency.

In addition to repairing the inconsistency, you might want to occasionally verify that the data is synchronized.

Enterprise Replication has tools that perform a full data synchronization (**`cdr sync`**), check data with repair (**`cdr check --repair`**), check data and report (**`cdr check`**), and repair specific failed transactions or rows (**`cdr repair ats`** and **`cdr repair ris`**).

The full data synchronization can be invoked either as part of the **`start replicate`** or **`start replset`** commands. Also, a full data synchronization can be performed as part of a stand-alone process. This process can be performed either from OAT or by using the cdr utility.

The check and synchronization can be performed either from OAT or from the command line. Also, it can be performed on a single replicated table or on a replicate set. Finally, it can be performed on any number of servers at the same time.

Note: We recommend that you select `replset` over `repl` when it comes to synchronizing tables that are involved in a referential relationship. If you group all the replicates of the tables that are in a referential relationship and use synchronization commands related to `replset`, then ER orders the synchronization of those tables in accordance with their parent-child relationship, thus causing fewer errors during the apply of the rows at the target node.

5.15.1 Performing a full synchronization

A full synchronization is normally done when starting a replicate or realizing a replicate set. However, it can also be performed on demand. Because it is normally done as part of the starting of a replicate or realizing a replicate set, there are options on those commands to automatically include a synchronization.

Using OAT to perform a full synchronization

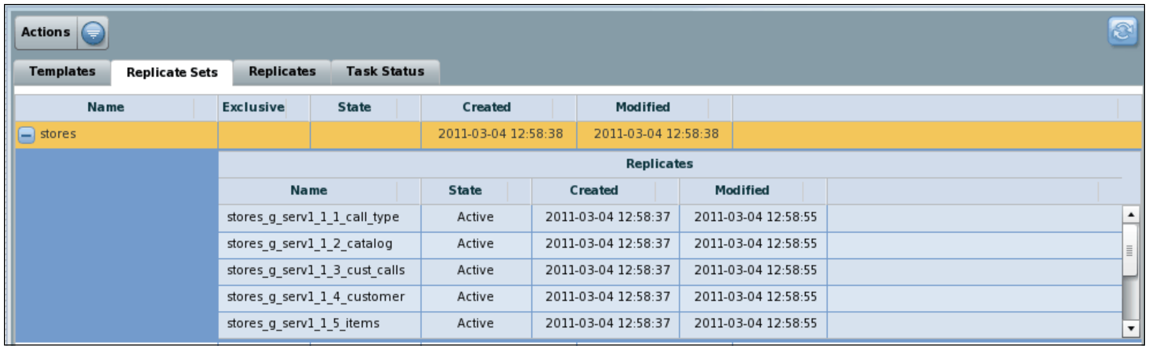
You can perform a synchronization on a single replicated table or on a replicate set. When you perform a synchronization on a replicate set, you determine the order of the tables to be synchronized based on any referential rules that have been defined on those tables. Synchronize the parent tables prior to synchronizing the child tables.

In this example, we assume that we have three servers, (serv1, serv2, serv3) and that the stores_demo database had been created on each of those servers, and that we used a template to define replication on the tables within the stores_demo database. Because we used a template name "stores" to create the replication, we also have a replicate set name "stores" that consists of all of the tables within the stores template. We now perform a synchronization of the replicate set using serv1 as a reference.

Note: We require a reference to identify a server that we consider to be correct. Synchronization involves copying the data from that node to the other nodes.

To synchronize the stores replicate set, complete the following steps:

1. Navigate to the replicate set page by selecting **Replication** → **Replicates** → **Replicate Sets** in OAT. Highlight the stores entry, as shown in Figure 5-53.



Name	Exclusive	State	Created	Modified
stores			2011-03-04 12:58:38	2011-03-04 12:58:38

Replicates

Name	State	Created	Modified
stores_g_serv1_1_1_call_type	Active	2011-03-04 12:58:37	2011-03-04 12:58:55
stores_g_serv1_1_2_catalog	Active	2011-03-04 12:58:37	2011-03-04 12:58:55
stores_g_serv1_1_3_cust_calls	Active	2011-03-04 12:58:37	2011-03-04 12:58:55
stores_g_serv1_1_4_customer	Active	2011-03-04 12:58:37	2011-03-04 12:58:55
stores_g_serv1_1_5_items	Active	2011-03-04 12:58:37	2011-03-04 12:58:55

Figure 5-53 Performing a synchronization of the stores replicate set

2. Open the **Actions** drop-down menu (Figure 5-54) and select **Check or Sync Replicate Set**.

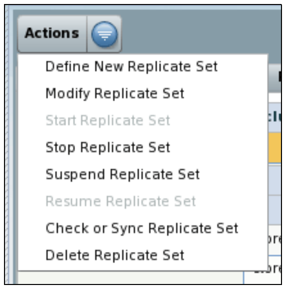
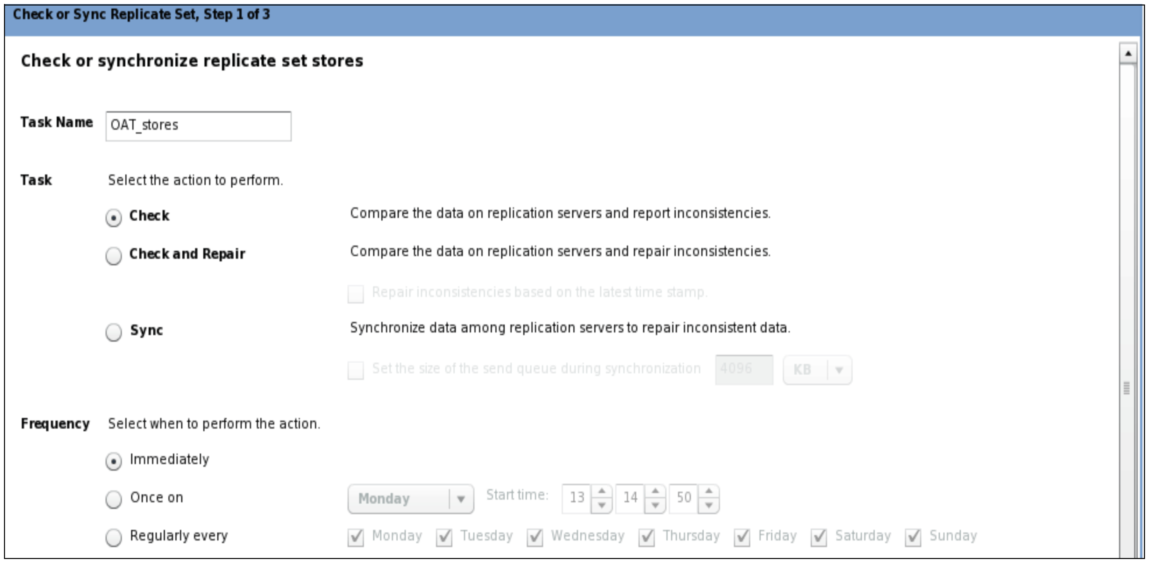


Figure 5-54 Action drop-down menu for replicate sets

3. The Check or Sync Replicate Set, Step 1 of 3 window opens (Figure 5-55).



Check or Sync Replicate Set, Step 1 of 3

Check or synchronize replicate set stores

Task Name: OAT_stores

Task: Select the action to perform.

- [✓ Check] Compare the data on replication servers and report inconsistencies.
- Check and Repair: Compare the data on replication servers and repair inconsistencies.
  - ☐ Repair inconsistencies based on the latest time stamp.
- Sync: Synchronize data among replication servers to repair inconsistent data.
  - ☐ Set the size of the send queue during synchronization 4096 KB

Frequency: Select when to perform the action.

- [✓ Immediately]
- Once on: Monday Start time: 13 14 50
- Regularly every: ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sunday

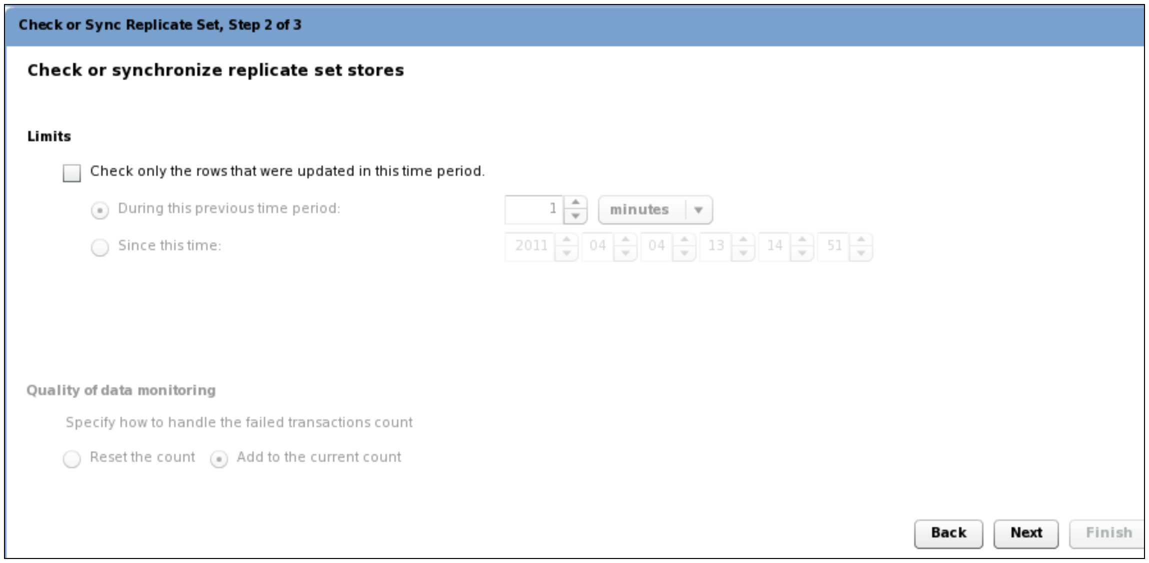
Figure 5-55 Main check/sync page for replicate set

From this window, you can perform a basic check, check with repair, and a synchronization. You also can make a scheduled task that could be repeatedly executed. If you choose to make a task, then this task will be inserted into the Informix dbcron tables. In our case, we execute the synchronization immediately. The Task Name is used for tracking the process and we use the default. Click **Next**.

4. In the Check or Sync Replicate Set, Step 2 of 3 window (Figure 5-56), you can limit the range of the synchronization to only consider changes made within a time range. You also can reset the quality of data row failure counts to zero.

   The Quality of Data (QOD) is used by the Connection Manager to determine if this server is a good choice for use based on latency and the number of failures in the apply. You can chose to clear the failure counters as part of the sync or check process.

   We do not have to select anything on this page so we simply click **Next**.


Check or Sync Replicate Set, Step 2 of 3

Check or synchronize replicate set stores

Limits

☐ Check only the rows that were updated in this time period.

[✓ During this previous time period:] 1 minutes

Since this time: 2011 04 04 13 14 51

Quality of data monitoring

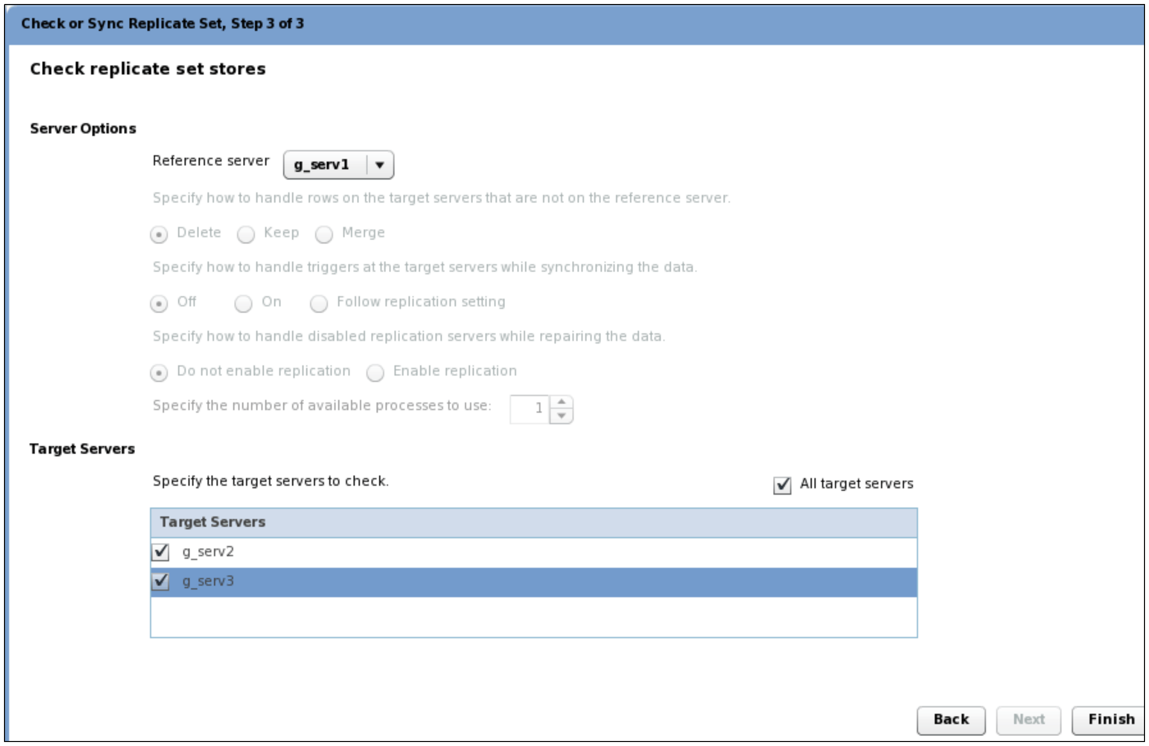
Specify how to handle the failed transactions count

Reset the count [✓ Add to the current count]

Figure 5-56 Limiting the check/sync

5. There are several options on the Check or Sync Replicate Set, Step 3 of 3 window (Figure 5-57). Because this is a synchronization and not a check, all the options are disabled. The only option you can select is the target servers. One server is chosen as the *Reference server* and the other servers are the targets. Changing the reference node changes the target servers automatically.

   In our case, we simply click **Finish** to perform the synchronization.


Check or Sync Replicate Set, Step 3 of 3

Check replicate set stores

Server Options

Reference server: g_serv1

Specify how to handle rows on the target servers that are not on the reference server.

[✓ Delete] Keep Merge

Specify how to handle triggers at the target servers while synchronizing the data.

[✓ Off] On Follow replication setting

Specify how to handle disabled replication servers while repairing the data.

[✓ Do not enable replication] Enable replication

Specify the number of available processes to use: 1

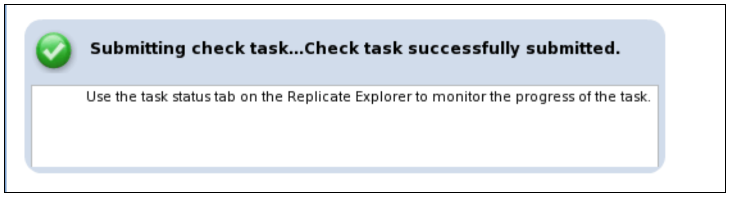
Target Servers

Specify the target servers to check. ✓ All target servers

Target Servers
✓ g_serv2
✓ g_serv3

Figure 5-57 Final window of check/sync

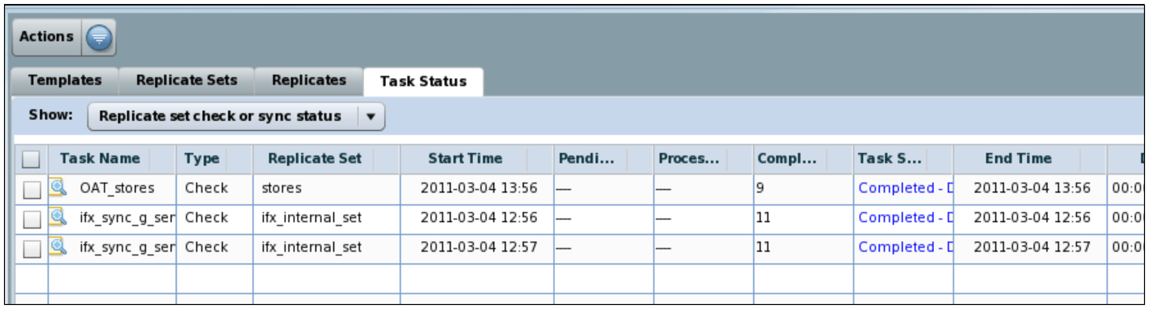
Figure 5-58 shows that the check task is submitted.


Submitting check task...Check task successfully submitted.

Use the task status tab on the Replicate Explorer to monitor the progress of the task.

Figure 5-58 Submission of sync/check

6. To view that status of the synchronization operation, we go to Task Status tab by selecting **Replicate** → **Replicates** → **Task Status** and select **Replicate set check or sync status** (Figure 5-59).

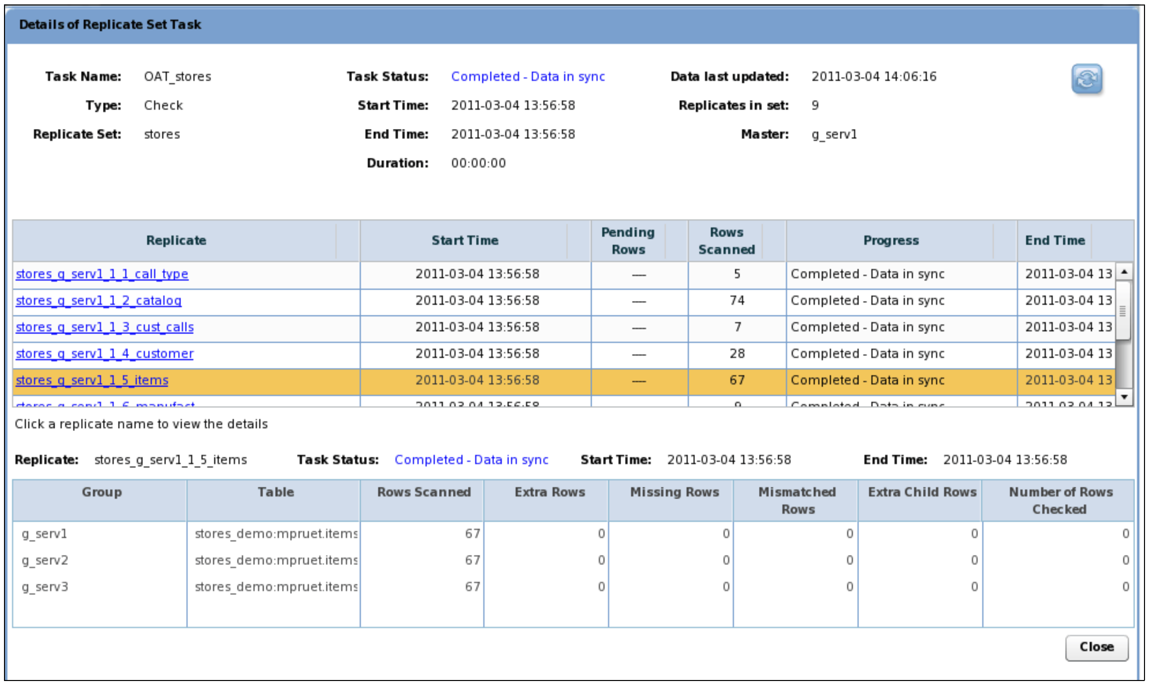


Show: Replicate set check or sync status

	Task Name	Type	Replicate Set	Start Time	Pendi...	Proces...	Compl...	Task S...	End Time	
☐	OAT_stores	Check	stores	2011-03-04 13:56	—	—	9	Completed - D	2011-03-04 13:56	00:0
☐	ifx_sync_g_ser	Check	ifx_internal_set	2011-03-04 12:56	—	—	11	Completed - D	2011-03-04 12:56	00:0
☐	ifx_sync_g_ser	Check	ifx_internal_set	2011-03-04 12:57	—	—	11	Completed - D	2011-03-04 12:57	00:0

Figure 5-59 Summary results of the synchronization of stores replicate set

7. To get detailed information about the synchronization, click the lookup icon ( ) in front of the task and select one of the replicates. For example, we look up OAT_store task and click stores_g_serv1_1_5_items to show its details (Figure 5-60).



Task Name:	OAT_stores	Task Status:	Completed - Data in sync	Data last updated:	2011-03-04 14:06:16
Type:	Check	Start Time:	2011-03-04 13:56:58	Replicates in set:	9
Replicate Set:	stores	End Time:	2011-03-04 13:56:58	Master:	g_serv1
		Duration:	00:00:00		

Replicate	Start Time	Pending Rows	Rows Scanned	Progress	End Time
stores_g_serv1_1_1_call_type	2011-03-04 13:56:58	—	5	Completed - Data in sync	2011-03-04 13
stores_g_serv1_1_2_catalog	2011-03-04 13:56:58	—	74	Completed - Data in sync	2011-03-04 13
stores_g_serv1_1_3_cust_calls	2011-03-04 13:56:58	—	7	Completed - Data in sync	2011-03-04 13
stores_g_serv1_1_4_customer	2011-03-04 13:56:58	—	28	Completed - Data in sync	2011-03-04 13
stores_g_serv1_1_5_items	2011-03-04 13:56:58	—	67	Completed - Data in sync	2011-03-04 13

Click a replicate name to view the details

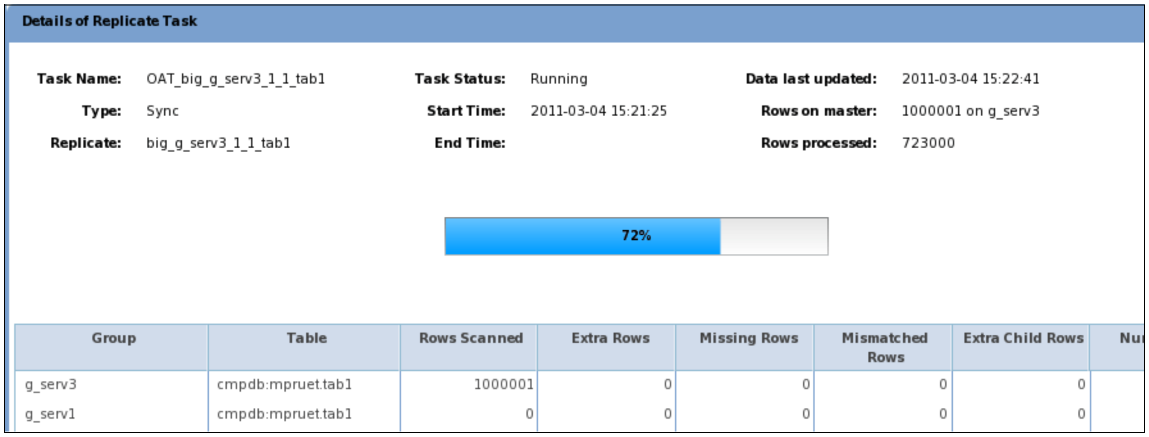
Replicate: stores_g_serv1_1_5_items Task Status: Completed - Data in sync Start Time: 2011-03-04 13:56:58 End Time: 2011-03-04 13:56:58

Group	Table	Rows Scanned	Extra Rows	Missing Rows	Mismatched Rows	Extra Child Rows	Number of Rows Checked
g_serv1	stores_demo:mpruet.items	67	0	0	0	0	0
g_serv2	stores_demo:mpruet.items	67	0	0	0	0	0
g_serv3	stores_demo:mpruet.items	67	0	0	0	0	0

Figure 5-60 Detailed results of synchronization of stores replicate set

Examining the progress indicator

It is possible to examine the progress of a synchronization or check it by navigating to the status detail page (select **Replication** → **Replicates** → **Task Status**) and then clicking the icon in front of the Task Name. An example is shown in Figure 5-61. In this example, we see that the synchronization is 72% completed.



Details of Replicate Task

Task Name:	OAT_big_g_serv3_1_1_tab1	**Task Status:**	Running	**Data last updated:**	2011-03-04 15:22:41
Type:	Sync	**Start Time:**	2011-03-04 15:21:25	**Rows on master:**	1000001 on g_serv3
Replicate:	big_g_serv3_1_1_tab1	**End Time:**		**Rows processed:**	723000

Group	Table	Rows Scanned	Extra Rows	Missing Rows	Mismatched Rows	Extra Child Rows	Nu
g_serv3	cmpdb:mpruet.tab1	1000001	0	0	0	0	
g_serv1	cmpdb:mpruet.tab1	0	0	0	0	0	

Figure 5-61 Viewing the progress indicator for a synchronization process

Using the cdr utility to execute a synchronization

The cdr utility can also be used to execute the synchronization of a replicate set or of a individual replicate. The synchronization can be performed either as a foreground or a background request. If the synchronization is performed as a background task, then an entry is made in the syscdr database and the server will manage the running of the cdr synchronization as a spawned task. You can still monitor the progress of the synchronization process by “naming” the task.

As an example, consider the foreground task shown in Example 5-58 to synchronize the stores replicate set.

Example 5-58 Synchronizing a replicate using cdr as a foreground task

```
cdr sync set --replset=stores --master=g_serv1 g_serv2

Mar 04 2011 19:56:36 ------   Table scan for stores_g_serv1_1_8_state start  --------
Starting synchronization scan for replicate
         stores_g_serv1_1_8_state

Ending synchronization scan for replicate
         stores_g_serv1_1_8_state
Ending synchronization scan for replicate
         stores_g_serv1_1_8_state
Mar 04 2011 19:56:38 ------   Table scan for stores_g_serv1_1_8_state end   ---------
Mar 04 2011 19:56:38 ------   Table scan for stores_g_serv1_1_4_customer start  --------
Starting synchronization scan for replicate
         stores_g_serv1_1_4_customer

Ending synchronization scan for replicate
         stores_g_serv1_1_4_customer
Ending synchronization scan for replicate
         stores_g_serv1_1_4_customer
Mar 04 2011 19:56:39 ------   Table scan for stores_g_serv1_1_4_customer end   ---------


Mar 04 2011 19:56:39 ------   Table scan for stores_g_serv1_1_6_manufact start  --------
Starting synchronization scan for replicate
         stores_g_serv1_1_6_manufact

Ending synchronization scan for replicate
         stores_g_serv1_1_6_manufact
Ending synchronization scan for replicate
         stores_g_serv1_1_6_manufact
Mar 04 2011 19:56:39 ------   Table scan for stores_g_serv1_1_6_manufact end   ---------
...
```

You can also view the progress indicator when using the cdr utility to perform a synchronization. The **cdr sync set** must be used for as a named task. Although this task can be done as a foreground synchronization, it probably makes more sense as a background task. Example 5-59 shows an example.

Example 5-59   Performing a synchronization using the cdr utility

```
cdr sync replset --replset=big --master=g_serv3 --name=BigSync g_serv1
--background
```

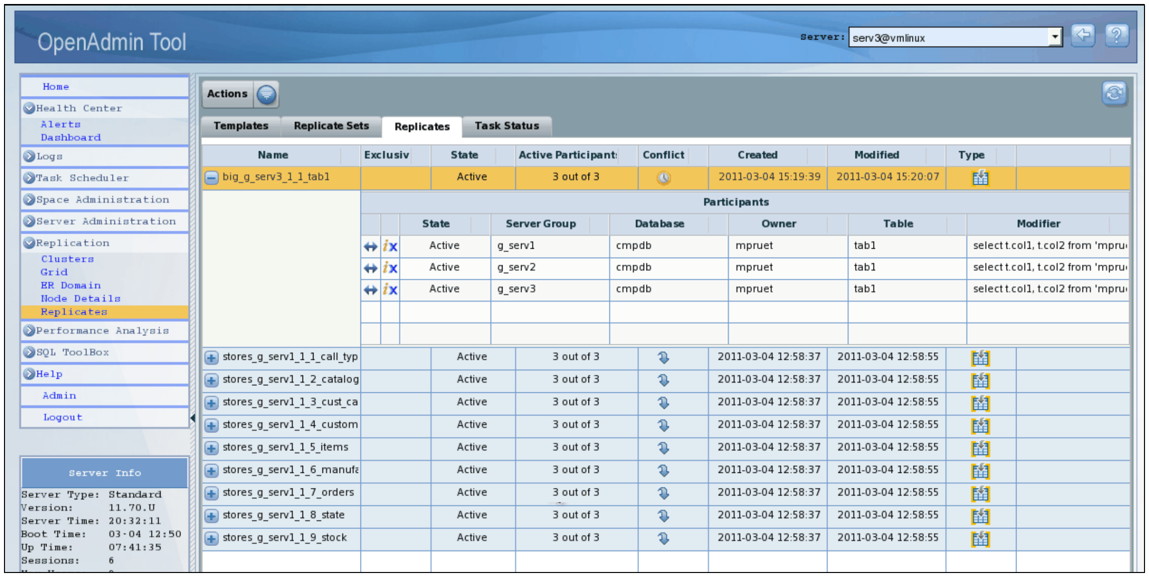
You can also look at the progress indicator by using the **cdr stats** command. This task requires that you “name” the **cdr sync replset** command and you must view the statistics of the synchronization from the server on which the command is executed. The statistics are not replicated to other nodes within the ER domain. In Example 5-60, the name of the synchronization is “BigSync”.

Example 5-60   Looking at the progress indicator from the command line

```
cdr stats sync
Task BigSync
big_g_serv3_1_1_tab1     Started Mar 04 20:10:17
*************-+----+----+----+----+----+----+----+ Remaining 0:00:22
```

Synchronization of a replicate

The procedure to synchronize a replicate is identical to the procedure to synchronize a replicate set. From OAT, instead of initially navigating to the replicate set page, navigate to the replicate page (select **Replication** → **Replicates** and select the **Replicates** tab). From there you can expand the replicate that we want to synchronize. In our case, we want to synchronize the big_g_serv3_1_1_tab1 replicate, so we highlight that replicate in yellow, as shown in Figure 5-62.



Server Info

Server Type:	Standard
Version:	11.70.U
Server Time:	20:32:11
Boot Time:	03-04 12:50
Up Time:	07:41:35
Sessions:	6

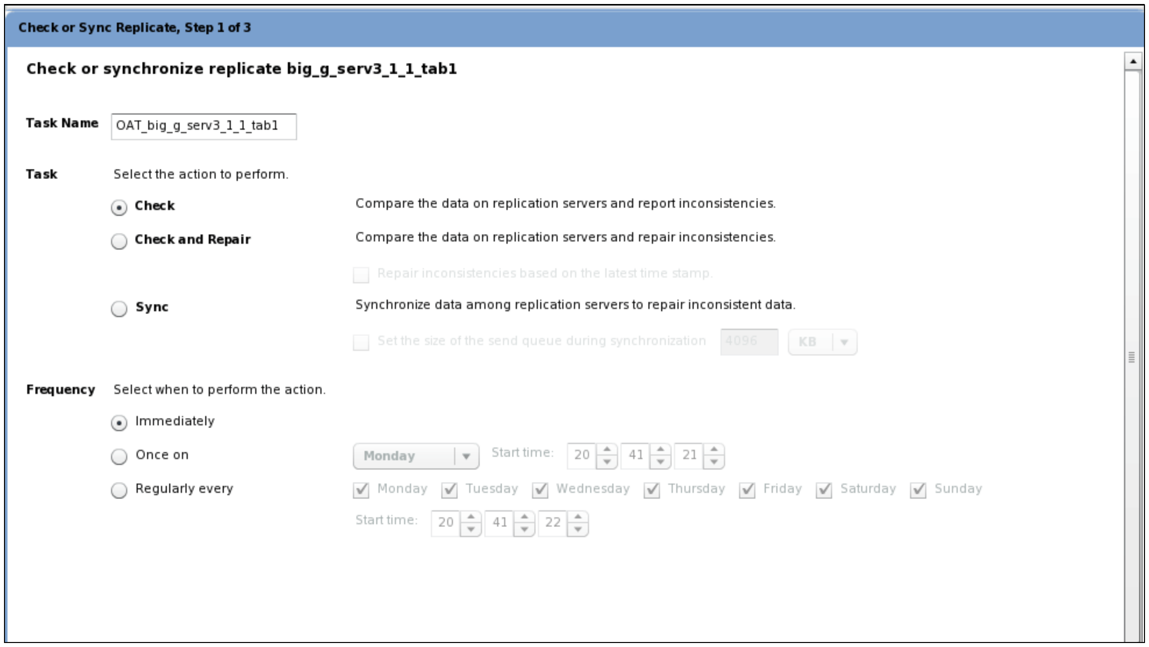
Name	Exclusiv	State	Active Participant:	Conflict	Created	Modified	Type
big_g_serv3_1_1_tab1		Active	3 out of 3		2011-03-04 15:19:39	2011-03-04 15:20:07	
stores_g_serv1_1_1_call_typ		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_2_catalog		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_3_cust_ca		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_4_custom		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_5_items		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_6_manufa		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_7_orders		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_8_state		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	
stores_g_serv1_1_9_stock		Active	3 out of 3		2011-03-04 12:58:37	2011-03-04 12:58:55	

Participants

State	Server Group	Database	Owner	Table	Modifier
Active	g_serv1	cmpdb	mpruet	tab1	select t.col1, t.col2 from 'mpru
Active	g_serv2	cmpdb	mpruet	tab1	select t.col1, t.col2 from 'mpru
Active	g_serv3	cmpdb	mpruet	tab1	select t.col1, t.col2 from 'mpru

Figure 5-62 The replicate page

Click the **Actions** drop-down menu and select **Check or Sync Replicate**. The Check or Sync Replicate, Step 1 of 3 window (Figure 5-63) opens.



Check or Sync Replicate, Step 1 of 3

Check or synchronize replicate big_g_serv3_1_1_tab1

Task Name OAT_big_g_serv3_1_1_tab1

Task Select the action to perform.

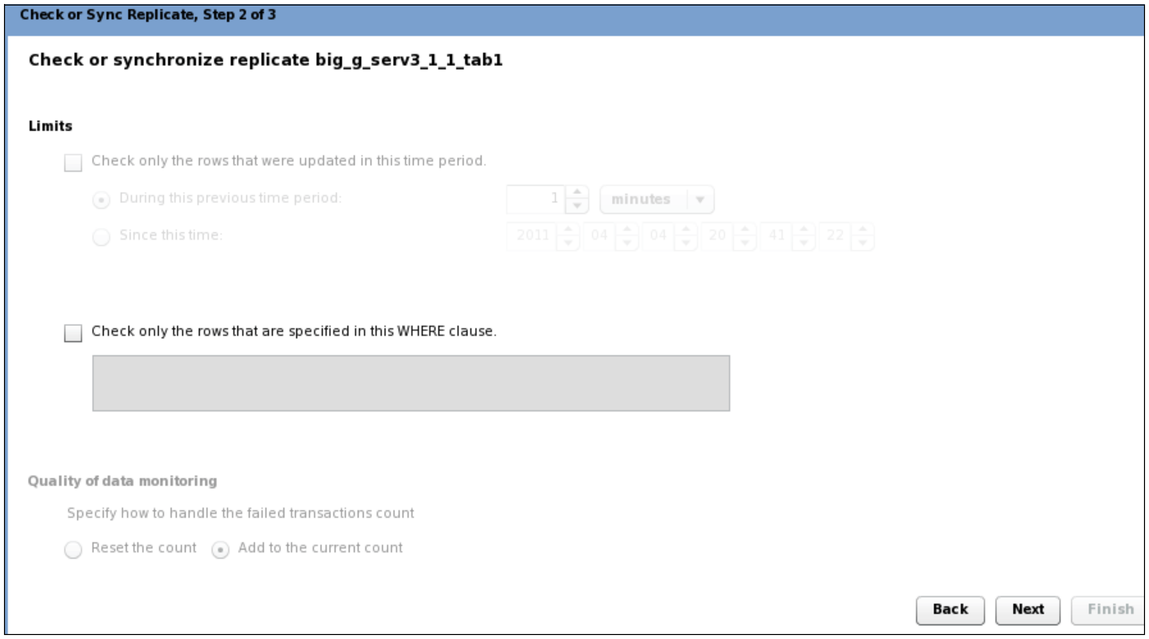
- [✓ **Check**] Compare the data on replication servers and report inconsistencies.
- **Check and Repair** Compare the data on replication servers and repair inconsistencies.
  - Repair inconsistencies based on the latest time stamp.
- **Sync** Synchronize data among replication servers to repair inconsistent data.
  - Set the size of the send queue during synchronization 4096 KB

Frequency Select when to perform the action.

- [✓ Immediately]
- Once on Monday Start time: 20 41 21
- Regularly every ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sunday
  Start time: 20 41 22

Figure 5-63 Performing a sync of a replicate

Select **Sync** and click **Next**. The Check or Sync Replicate, Step 2 of 3 window (Figure 5-64) opens, and is similar to the Sync Replicate Set, Step 2 of 3 window, but with a significant difference. This window has an added option for a WHERE clause to be added to the synchronization. This option allows us to restrict the synchronization to a subset of the rows, which is useful in the case where only a fragment of the rows within the table need to be synchronized. The WHERE option is only available n the replicate synchronization because it would not be possible to have a common restriction on a replicate set.


Check or Sync Replicate, Step 2 of 3

Check or synchronize replicate big_g_serv3_1_1_tab1

Limits

☐ Check only the rows that were updated in this time period.

During this previous time period: 1 minutes

Since this time: 2011 04 04 20 41 22

☐ Check only the rows that are specified in this WHERE clause.

Quality of data monitoring

Specify how to handle the failed transactions count

Reset the count [✓ Add to the current count]

Figure 5-64 Restricting the replicate synchronization

We do not need to include any additional restrictions so we can simply click **Next**. The Check or Sync Replicate, Step 3 of 3 window (Figure 5-57 on page 215) opens. We only want to synchronize between g_server1 and g_server2, so we can clear g_serv3 as a target and then click **Finish**.

Performing a synchronization of a replicate using cdr

Just as with the replicate set, you can perform a synchronization of an individual replicate using the cdr utility. Example 5-61 shows an example of using cdr to synchronize a replicate.

Example 5-61 Using the cdr utility to synchronize a table

```
cdr sync repl --repl=big_g_serv3_1_1_tab1 --master=g_serv1 g_serv2

Mar 04 2011 21:02:03 ------   Table scan for big_g_serv3_1_1_tab1 start ------
Starting synchronization scan for replicate
          big_g_serv3_1_1_tab1

Ending synchronization scan for replicate
          big_g_serv3_1_1_tab1
Mar 04 2011 21:03:58 ------   Table scan for big_g_serv3_1_1_tab1 end --------
```

5.15.2 Starting a replicate with an initial synchronization

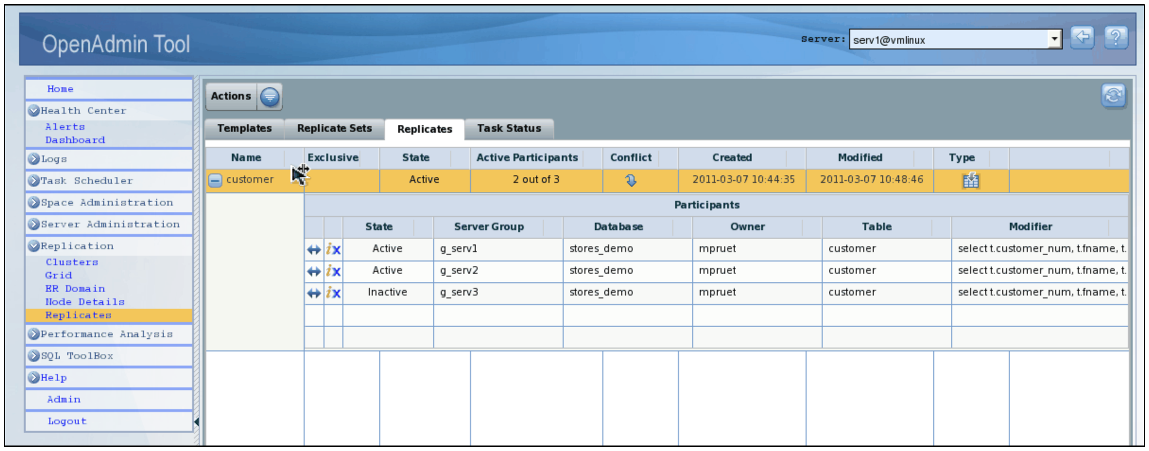
When you start a replicate, we should have an initial synchronization of the data. Otherwise, you might be starting with inconsistent data. The start of a replicate must always be performed when you initially define a replicate or when you add a new server to an existing replicate.

Note that the template does not have a specific start; rather, the replicate is automatically started when the template is realized on a server. You also can perform an initial synchronization as part of the realize template.

Starting a replicate with initial synchronization using OAT

For this example, we assume that we have a "customer" replicate that exists on the customer table from the stores_demo database. We have three servers within the Enterprise Replication domain, serv1, serv2, and serv3. We have already established replication between serv1 and serv2, but are now adding replication to serv3. We have already added serv3 as a member of the customer replicate, but have not yet started replication on that node.

To start the replicate, we navigate to the replicate page by selecting **Replicate** → **Replicates** and then selecting the **Replicates** tab. When we expand the customer replicate, we notice that the replicate is active on serv1 and serv2, but is inactive on serv3. We double-click the customer row to select serv3 as the object of the start action (Figure 5-65).



Name	Exclusive	State	Active Participants	Conflict	Created	Modified	Type
customer		Active	2 out of 3		2011-03-07 10:44:35	2011-03-07 10:48:46	

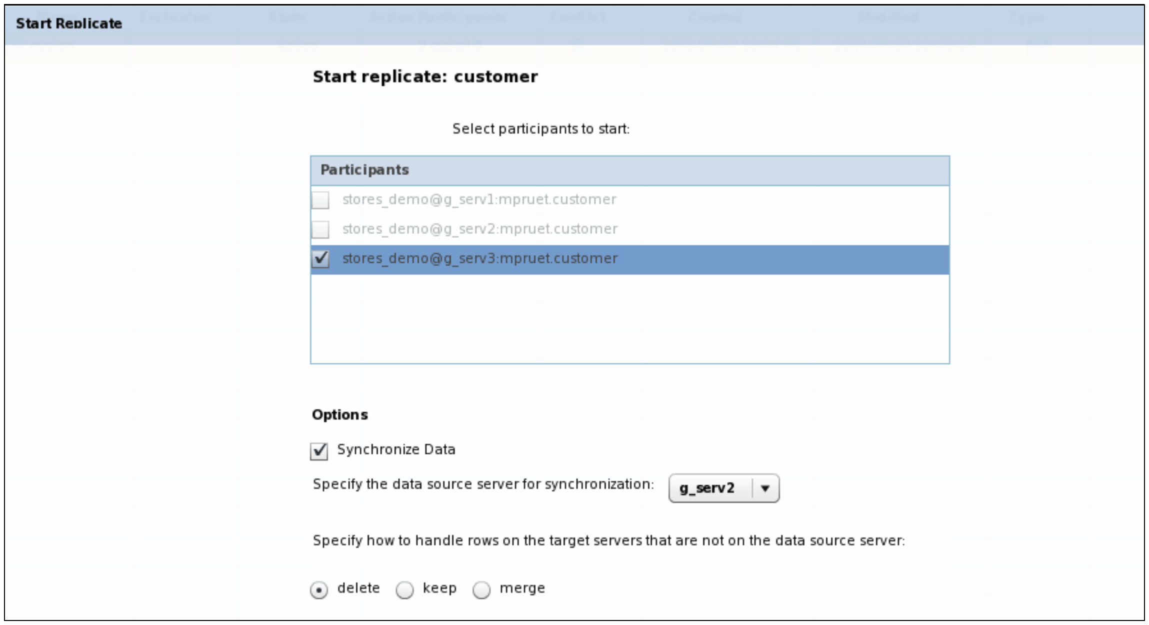
Participants

State	Server Group	Database	Owner	Table	Modifier
Active	g_serv1	stores_demo	mpruet	customer	select t.customer_num, t.fname, t.
Active	g_serv2	stores_demo	mpruet	customer	select t.customer_num, t.fname, t.
Inactive	g_serv3	stores_demo	mpruet	customer	select t.customer_num, t.fname, t.

Figure 5-65 Preparing to start the customer replicate on serv3

From the Actions drop-down menu, select **Start Replicate** and the Start Replicate window opens. This window is used to choose on which server to start this replicate on and whether to perform an initial synchronization as part of the start. The Synchronize Data gives an option to handle any extra rows encountered on the new server. You can delete them (default), keep them, or merge them. If merge is chosen, the extra rows on the new server are replicated to the other nodes within the replication domain.

We perform the synchronization with g_serv2 and keep the default “delete” option for the extra rows (Figure 5-66). Click **OK** to start the replicate and perform the initial synchronization.


Start Replicate

Start replicate: customer

Select participants to start:

Participants
☐ stores_demo@g_serv1:mpruet.customer
☐ stores_demo@g_serv2:mpruet.customer
✓ stores_demo@g_serv3:mpruet.customer

Options

✓ Synchronize Data

Specify the data source server for synchronization: g_serv2

Specify how to handle rows on the target servers that are not on the data source server:

[✓ delete] keep merge

Figure 5-66 Options for initial synchronization

Using the cdr utility to perform a start with synchronization

In this section, we use the same scenario described above to demonstrate how to start a replicate with synchronization using the cdr utility. The customer replicate is replicating between serv1 and serv2, but not serv3. We have added serv3 as a participant, but have not yet started it. If we examine the output of **`cdr list repl`** on serv1 and serv3, we notice that the replicate is active on serv1, but not on serv3. Example 5-62 and Example 5-63 on page 225 list the replicate in serv1 and serv3.

Example 5-62 cdr list repl on serv1

```
$cdr list repl

CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        customer
STATE:            Active ON:g_serv1
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
```

```
PARTICIPANT:      stores_demo:mpruet.customer
OPTIONS:          row,ris,ats,floatieee,fullrow
REPLID:           65539 / 0x10003
REPLMODE:         PRIMARY  ON:g_serv1
APPLY-AS:         INFORMIX ON:g_serv1
REPLTYPE:         Master
```

Example 5-63 cdr list repl on serv3

```
$cdr list repl

CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        customer
STATE:            Inactive ON:g_serv3
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores_demo:mpruet.customer
OPTIONS:          row,ris,ats,floatieee,fullrow
REPLID:           65539 / 0x10003
REPLMODE:         PRIMARY  ON:g_serv3
APPLY-AS:         INFORMIX ON:g_serv3
REPLTYPE:         Master
```

Example 5-64 shows the **cdr** command to start the replicate and the command's output. We request a synchronization by including the --syncdatasource option. Also, we request a foreground synchronization so that we can see the results.

Example 5-64 Starting a replicate with synchronization in foreground using cdr

```
cdr start repl customer --syncdatasource=g_serv2 \
--extratargetrows=delete --foreground g_serv3

Mar 07 2011 11:19:08 ------   Table scan for customer start  --------
Starting synchronization scan for replicate
          customer

Ending synchronization scan for replicate
          customer
Mar 07 2011 11:19:09 ------   Table scan for customer end   ---------
```

5.15.3 Checking if the tables are synchronized

In addition to performing an initial synchronization, we want to be able to periodically verify that the data is still synchronized. As part of this task, we also want the verification to automatically repair any differences that it encounters. We want not only to do this on a single replicate, but on a replicate set. Also, we want to be able to perform the check on multiple servers.

When performing a check with repair, any rows that are not the same at the time of the check are repaired automatically. If there is an extra row on the target machine, it is detected. Any row that is the child of a referential integrity relationship with that row is also detected. When performing a check of a replicate set, the order the tables is checked follows the referential integrity rules. The parent tables are checked before the child rows.

You might want to use the check with repair instead of the initial synchronization. For example, you can use high performance load (HPL) or external tables to create a copy of the data to be used to populate a new replicate. If the source of the data is idle, then this would result in a consistent copy of the data. But if the source is active, then the resulting copy of the data would be inconsistent. You can be 90% consistent and use check with repair to resolve these inconsistencies (Figure 5-67).

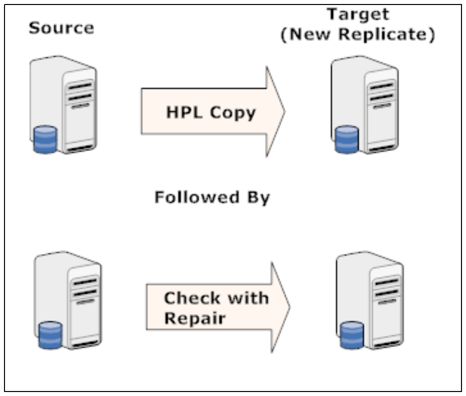


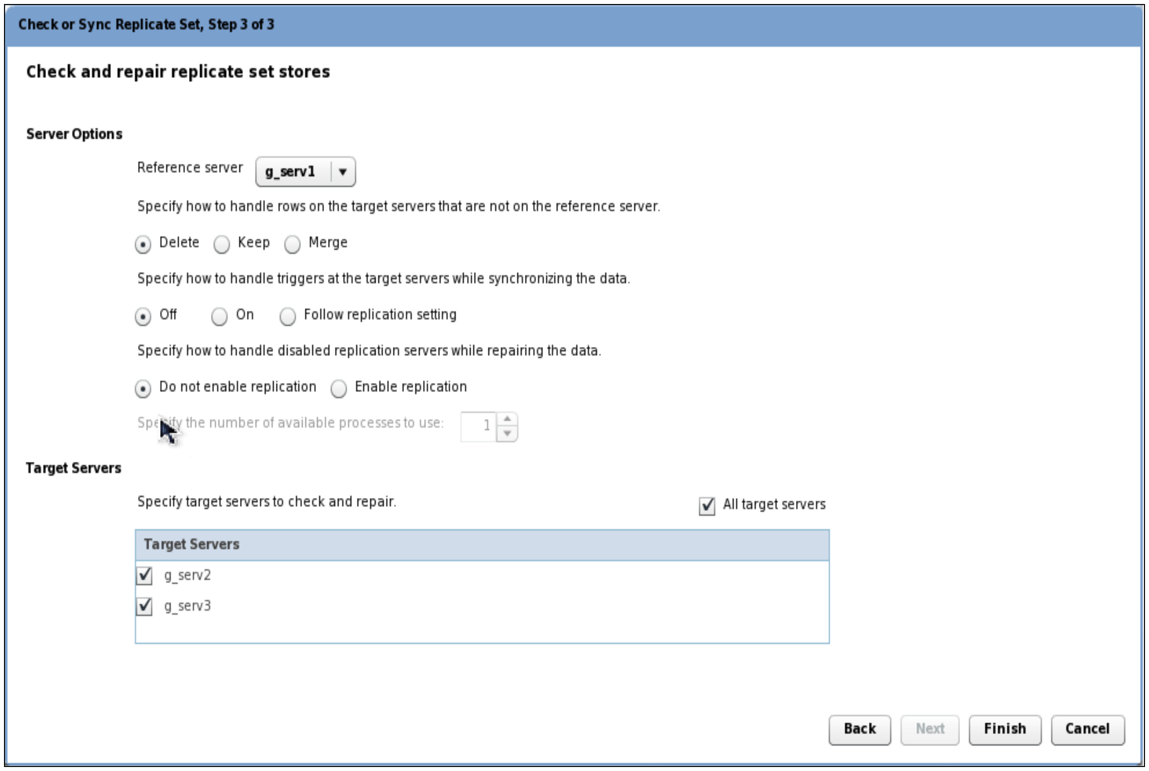
Figure 5-67 Using repair check to perform an initial synchronization

Performing a replicate check using OAT

There is little difference between a full synchronization, a check, and a check with repair. The process to perform each of these tasks is pretty much the same. The only significant difference is the outcome. A synchronization (`cdr sync`) repairs everything, a check with repair only repairs the rows that it needs to repair, and a basic check only reports on the differences.

To demonstrate the steps of checking replicate using OAT, as before, we use three servers, serv1, serv2, and serv3. We use cdr1 as our reference server.

To perform a check using OAT, navigate to the replicate or replicate set window. In our case, we check the stores replicate set that consists of tables that are part of the stores database, so we navigate to the replicate set window by selecting **Replicate** → **Replicates** and selecting the **Replicate Sets** tab. In this example, we select **Check and Repair** and click **Next** until we get to the submission window (Figure 5-68).



Check and repair replicate set stores

Server Options

Reference server: g_serv1

Specify how to handle rows on the target servers that are not on the reference server.

[✓ Delete] Keep Merge

Specify how to handle triggers at the target servers while synchronizing the data.

[✓ Off] On Follow replication setting

Specify how to handle disabled replication servers while repairing the data.

[✓ Do not enable replication] Enable replication

Specify the number of available processes to use: 1

Target Servers

Specify target servers to check and repair.

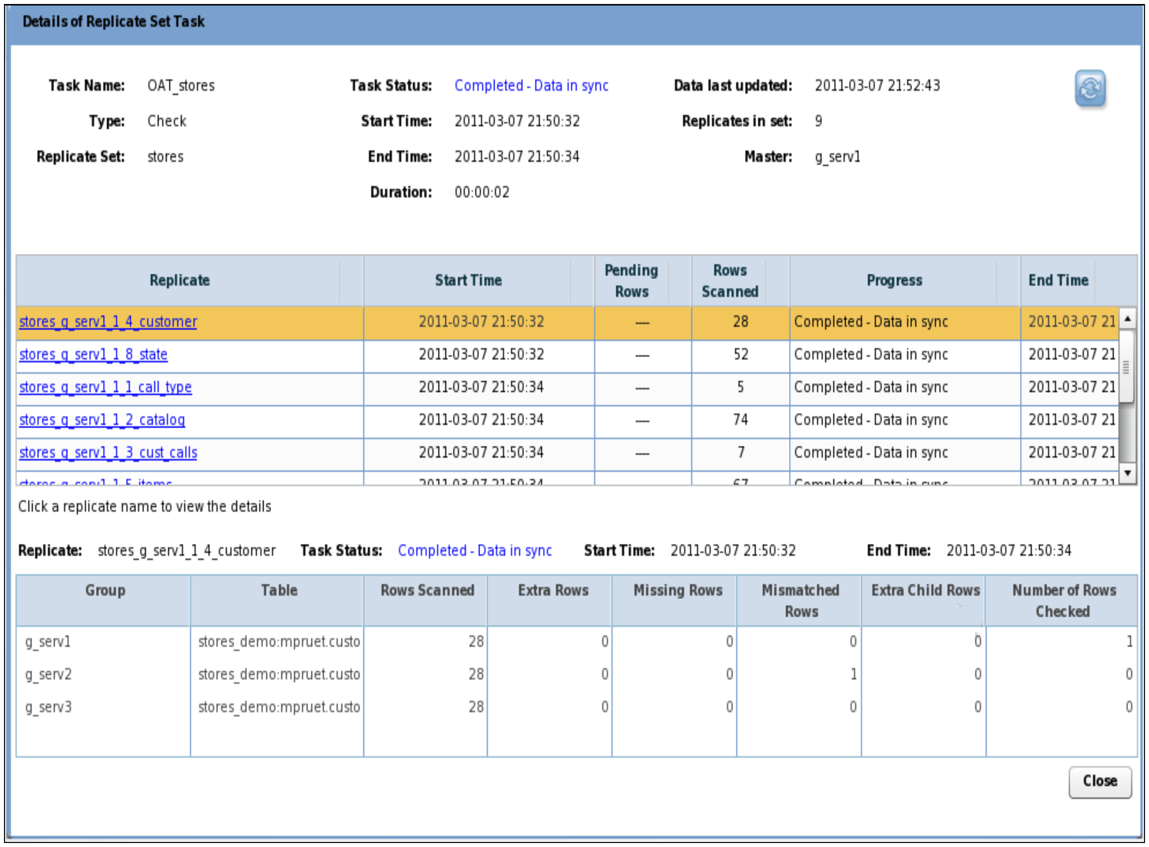
✓ All target servers

Target Servers
✓ g_serv2
✓ g_serv3

Figure 5-68 Check submission window

Again, we chose to use the defaults and click **Finish**. This action submits the check for the stores replicate set. If any rows are out of sync, the check with repair corrects the problem, and then verifies that the repair was successfully performed.

After submitting the check with repair, we examine the completion of the task by selecting **Task Status** and then selecting the icon in front of the OAT_stores entry. Then we select the stores_g_serv1_1_customer entry. (Figure 5-69).



Details of Replicate Set Task

Task Name:	OAT_stores	**Task Status:**	Completed - Data in sync	**Data last updated:**	2011-03-07 21:52:43
Type:	Check	**Start Time:**	2011-03-07 21:50:32	**Replicates in set:**	9
Replicate Set:	stores	**End Time:**	2011-03-07 21:50:34	**Master:**	g_serv1
		Duration:	00:00:02		

Replicate	Start Time	Pending Rows	Rows Scanned	Progress	End Time
stores_g_serv1_1_4_customer	2011-03-07 21:50:32	—	28	Completed - Data in sync	2011-03-07 21
stores_g_serv1_1_8_state	2011-03-07 21:50:32	—	52	Completed - Data in sync	2011-03-07 21
stores_g_serv1_1_1_call_type	2011-03-07 21:50:34	—	5	Completed - Data in sync	2011-03-07 21
stores_g_serv1_1_2_catalog	2011-03-07 21:50:34	—	74	Completed - Data in sync	2011-03-07 21
stores_g_serv1_1_3_cust_calls	2011-03-07 21:50:34	—	7	Completed - Data in sync	2011-03-07 21
stores_g_serv1_1_5_items	2011-03-07 21:50:34		67	Completed - Data in sync	2011-03-07 21

Click a replicate name to view the details

Replicate: stores_g_serv1_1_4_customer **Task Status:** Completed - Data in sync **Start Time:** 2011-03-07 21:50:32 **End Time:** 2011-03-07 21:50:34

Group	Table	Rows Scanned	Extra Rows	Missing Rows	Mismatched Rows	Extra Child Rows	Number of Rows Checked
g_serv1	stores_demo:mpruet.custo	28	0	0	0	0	1
g_serv2	stores_demo:mpruet.custo	28	0	0	1	0	0
g_serv3	stores_demo:mpruet.custo	28	0	0	0	0	0

Figure 5-69 Check repair of stores replicate set

We can see from this example that there was one row out of sync on serv2 and that out of sync row causes a row on serv1 to be replicated.

Performing a check from the cdr utility

The cdr utility can be used to perform a check replicate or a replicate set as well. The check of a single replicate is performed by the **`cdr check repl`** command, and the check of a replicate set is performed by the **`cdr check set`** command. We can also perform a repair by adding --repair as an option to the commands. Example 5-65 shows an example of how to perform the check repair of the stores replicate set.

Example 5-65 Checking a replicate set using the cdr utility

```
$cdr check set --replset=stores --master=g_serv1 --repair g_serv2 g_serv3

Mar 07 2011 22:05:18 ------   Table scan for stores_g_serv1_1_8_state start  --------

Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
g_serv2                 52         0         0         0         0
g_serv3                 52         0         0         0         0


Mar 07 2011 22:05:18 ------   Table scan for stores_g_serv1_1_8_state end   ---------


Mar 07 2011 22:05:18 ------   Table scan for stores_g_serv1_1_4_customer start  --------

Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 28         0         0         0         1
g_serv2                 28         0         0         1         0
g_serv3                 28         0         0         0         0


The repair operation completed. Validating the repaired rows ...
Validation completed successfully.
Mar 07 2011 22:05:18 ------   Table scan for stores_g_serv1_1_4_customer end   ---------


Mar 07 2011 22:05:18 ------   Table scan for stores_g_serv1_1_6_manufact start  --------

Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                  9         0         0         0         0
g_serv2                  9         0         0         0         0
g_serv3                  9         0         0         0         0

...
```

We see that the customer table discovered that there was a row that did not match on serv2, which caused the row on serv1 to be replicated. We also notice that we verified the repaired rows and that the validation was successful.

Using --timestamp for reference

The **cdr check** command has some options that are not currently available from OAT. One of these is *--timestamp*.

Normally, when performing a check or a check repair, we require that one server act as a reference. This server is identified by the --master option. This server is used as the "correct" server and is the server to which the other nodes will become consistent.

However, that server is not always the correct choice for the reference. For example, it might be that for some rows server-A should act as the reference and for other rows Server-B might be a better choice. As an example, consider Figure 5-70.

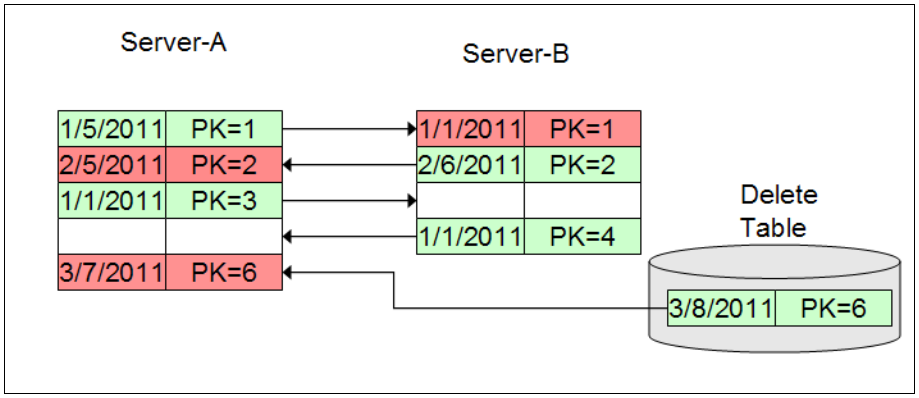


Figure 5-70 Using --timestamp with check repair

In this example, there are two servers that we are performing a check repair and have chosen to use the --timestamp option. In the row where the primary key is equal to 1, the row on Server-A has the most recent time stamp, so for that row, Server-A is used as the reference to synchronize Server-B with the row on Server-A. However, for the row with the primary key equal to 2, the row on Server-B is the more current, so Server-B is used as the reference row. For the row where the primary key is 3, there is no row on Server-B, so Server-A is the reference and the row on Server-A is propagated to server-B.

The row with the primary key is 6 is a special case. The row exists in the table on Server-A, but not on Server-B. The delete table on Server-B is then checked Because not only does the row exist in the delete table on Server-B, but it is also a more current row, Server-B is used as the reference node for that row and the row is deleted on Server-A.

To support the --timestamp option, you have to delay the purge of the rows from the delete tables by setting the **onconfig** parameter CDR_DELAY_PURGE_DTC to the amount of time that you want to delay the purging of rows from the delete tables. The value can be set to the number of seconds, minutes, hours, or days that the rows will be retained in the delete tables after you have normally purged them. As an example, setting `CDR_DELAY_PURGE_DTC        7D` specifies that you want to retain rows in the delete tables for 7 days after you have normally purged those rows.

5.15.4 Using cdr repair ats

If the replicate is defined with an ATS or RIS attribute, then you can restrict the repair to the specific rows contained within those files. This action significantly reduces the time that it takes to perform a repair of systems that are not consistent.

As an example, let us define a replicate on the customer table in the stores_demo database that is using ignore conflict resolution. We chose ignore conflict resolution because that choice for conflict resolution requires that the row be applied with exactly the same operation as was done on the source. An update is not automatically converted into an insert if the row is missing on one of the targets.

If we perform a check of the customer table, we get the results shown in Example 5-66.

Example 5-66   Check of the customer replicate

```
cdr check repl --replicate=customer --master=g_serv1 --all

Mar 07 2011 22:41:19 ------   Table scan for customer start  --------

Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 28         0         0         0         0
g_serv2                 28         0         0         0         0
g_serv3                 27         0         1         0         0

WARNING: replicate is not in sync
Mar 07 2011 22:41:20 ------   Table scan for customer end   ---------

command failed -- WARNING: replicate is not in sync (178)
```

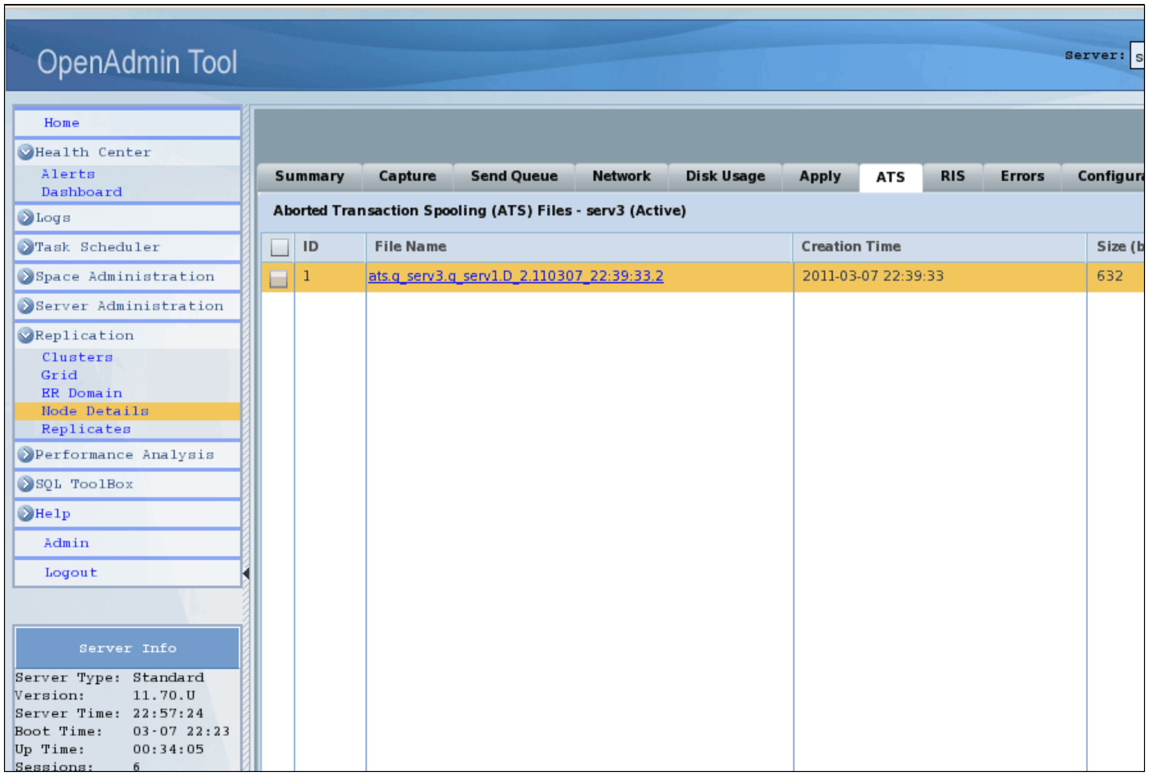
We check the ATS directory and see that we have a file that has been created as the result of a failure of the apply. Its contents indicate that a row did not exist on the target server (Example 5-67), and that caused the apply to fail.

Example 5-67 Contents of the ATS file

```
TXH RIS file:RIS/ris.g_serv3.g_serv1.D_2.110307_22:39:33.1 has also been
created for this transaction
==========
TXH Source ID:1 / Name:g_serv1 / CommitTime:11-03-07 22:39:32
TXH Target ID:3 / Name:g_serv3 / ReceiveTime:11-03-07 22:39:33
TXH Number of rows processed when transaction was aborted:1
TXH All rows in a transaction defined with row scope were rejected
TXH CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 /
ISAM:0
----------
RRH Row:1 / Replicate Id: 65539 / Table: stores_demo@mpruet.customer /
DbOp:Update
RRD 102|Carole|Stanford|Sports Spot|785 Geary St||San
Francisco|CA|94117|415-822-1289
```

Using OAT to repair the ATS errors

We can repair the rows contained in the ATS or RIS files by selecting serv3 from the **Server** drop-down menu, navigating to the ATS window by selecting **Replication** → **Node Details**, and then selecting the **ATS** tab (Figure 5-71).



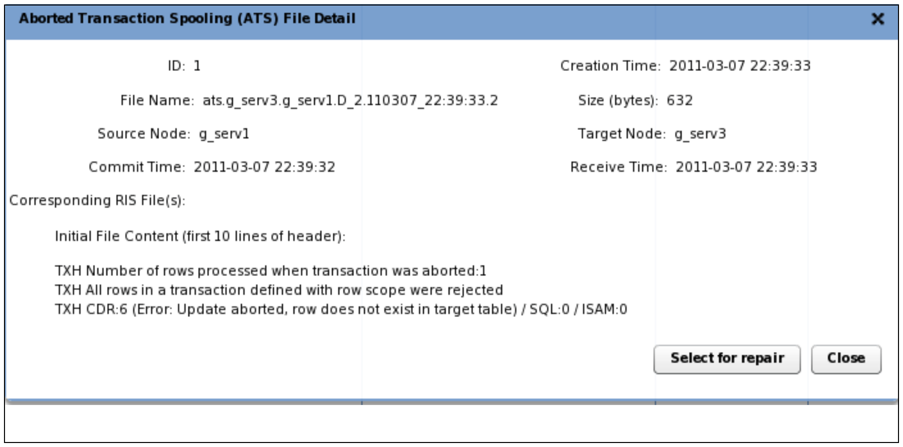
Server Type: Standard
Version: 11.70.U
Server Time: 22:57:24
Boot Time: 03-07 22:23
Up Time: 00:34:05
Sessions: 6

Aborted Transaction Spooling (ATS) Files - serv3 (Active)

	ID	File Name	Creation Time	Size (b
☐	1	ats.q_serv3.q_serv1.D_2.110307_22:39:33.2	2011-03-07 22:39:33	632

Figure 5-71 ATS window

If we select the ATS file, we get details about the error that was encountered (Figure 5-72).


Aborted Transaction Spooling (ATS) File Detail

ID: 1

Creation Time: 2011-03-07 22:39:33

File Name: ats.g_serv3.g_serv1.D_2.110307_22:39:33.2

Size (bytes): 632

Source Node: g_serv1

Target Node: g_serv3

Commit Time: 2011-03-07 22:39:32

Receive Time: 2011-03-07 22:39:33

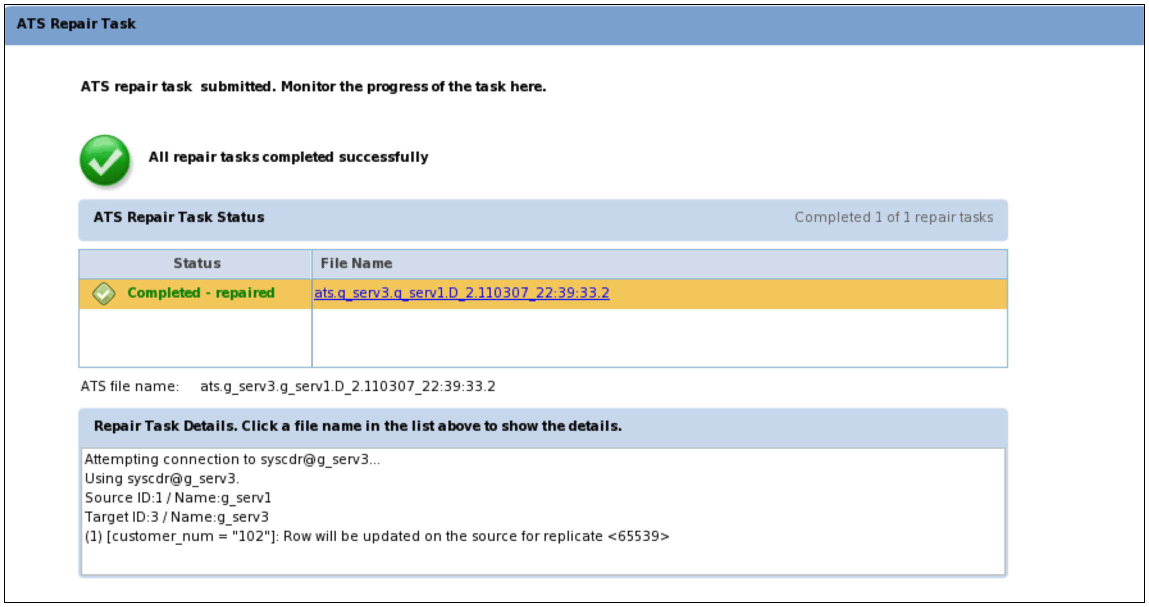
Corresponding RIS File(s):

Initial File Content (first 10 lines of header):

```
TXH Number of rows processed when transaction was aborted:1
TXH All rows in a transaction defined with row scope were rejected
TXH CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
```

Figure 5-72 Details of the ATS file

We click **Select** for repair followed by **Close** to select the file as a candidate for the ATS repair. From there, we click **Repair** to process the ATS file. After the repair is complete, a confirmation window opens (Figure 5-73).


ATS Repair Task

ATS repair task submitted. Monitor the progress of the task here.

All repair tasks completed successfully

ATS Repair Task Status — Completed 1 of 1 repair tasks

Status	File Name
Completed - repaired	ats.g_serv3.g_serv1.D_2.110307_22:39:33.2

ATS file name: ats.g_serv3.g_serv1.D_2.110307_22:39:33.2

Repair Task Details. Click a file name in the list above to show the details.

```
Attempting connection to syscdr@g_serv3...
Using syscdr@g_serv3.
Source ID:1 / Name:g_serv1
Target ID:3 / Name:g_serv3
(1) [customer_num = "102"]: Row will be updated on the source for replicate <65539>
```

Figure 5-73 Repair complete window

If we perform another check of the customer table, we discover that the table is now synchronized (Example 5-68).

Example 5-68 Check after the ATS repair

```
cdr check repl --replicate=customer --master=g_serv1 --all

Mar 07 2011 23:15:31 ------   Table scan for customer start  --------

Node                  Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                 28         0         0         0         0
g_serv2                 28         0         0         0         0
g_serv3                 28         0         0         0         0

Mar 07 2011 23:15:31 ------   Table scan for customer end   ---------
```

We check the ATS directory again and see that the file has been deleted to prevent it from being accidentally applied again.

Example 5-69 shows how to use the cdr utility to repair an ATS file.

Example 5-69 Repairing an ATS file using cdr

```
$cdr repair ats --verbose ats.g_serv3.g_serv1.D_3.110307_23:19:59.2
Attempting connection to syscdr@g_serv3...
Using syscdr@g_serv3.
Source ID:1 / Name:g_serv1
Target ID:3 / Name:g_serv3
(1) [customer_num = "102"]: Row will be updated on the source for replicate
<65539>
```

5.15.5 Improving the performance of sync and check

It can take a long time to synchronize or check a large set of data over multiple nodes. There are several options that can be used to improve the performance of a synchronization or a check of a replicate or of a replicate set.

Increasing the queue size

The queue size is a critical factor in the synchronization of a replicate or of a replicate set. During the synchronization process, the server does its best to avoid having to spool the send queue to disk because the cost of spooling to disk is expensive and has an overall negative impact on performance. Because of this situation, the server throttles the synchronization to avoid spooling the queue to stable storage.

However, in general, the amount of data that is placed in the queue is significantly higher during a synchronization process than during normal activity. Therefore, there is an advantage to increasing the maximum size that the queue can expand while **`cdr sync`** is processing.

You can define a temporary expansion of the queue memory size while running the synchronization by using the --memadjust parameter with the **`cdr sync replicate`** and **`cdr sync set`** commands. For example, you could expand the queue size by adding --memadjust=500M, which would expand the queue size to 500 MB for the duration of the synchronization.

Increasing parallelism

When processing a replicate set as part of a **`cdr sync set`** or **`cdr check set`** command, you can increase parallelism by using the --process option. This option allows the server to sync or check multiple tables at the same time, and thus can decrease the overall run time.

When performing the check or synchronization of a replicate set, the cdr utility separates the set of tables that we are going to process into groups based on referential integrity rules. It then processes the parent table groups before processing any table within the child group. Within each group, we then order the tables based on size (larger tables are processed first). We then can process each table within the groups in parallel.

If you define --process=6, Informix spawns six processes to perform the check or synchronization and each process processes an individual table within the current group.

Ignoring large objects

It can take a lot of time to verify that a large object is synchronized. You can specify that you do not want to check the large objects when performing a check of a replicate or of a replicate set. This is done by specifying --skipLOB in the check of a replicate or of a replicate set.

Generally, the large objects such as byte or BLOB columns are not changed during the life of the row, so all that needs to be checked is whether the large object exists.

Using the since option

If using time stamp conflict resolution, the time that it takes to perform a check of a large table can be reduced by only considering rows that were updated since the last time that a check was performed. For example, if a check was made on a large table on June 10, 2011, then you can reduce the scope of a subsequent check by specifying that you only want to consider rows that were changed since June 10, 2011. This task can be done from OAT or from the command line. The **cdr** command line option is --since=<time>, where time can be a date or a number of units in the past. For example, to check rows updated on or after June 10, 2011, specify --since=2011-06-10; to include rows that had been updated in the past two weeks, specify --since=2W.

Using the where option

In addition to the since option, the rows considered as part of the check of a large table can be reduced by adding an extra *where* clause. For example, suppose that you have a table that was fragmented by expression and one of the partitions became corrupted by, for example, a media failure. If you want to repair only that fragment, you can perform this action by adding the where option to the check.

The where clause simply appends the extra restriction to the replicate definition that is used during the check. It allows us to have flexibility in the reduction of the number of rows considered as part of a check. The where clause must be contained in quotes as a character string.

As an example, we could reduce the scope of a check of an employee table by running the following command:

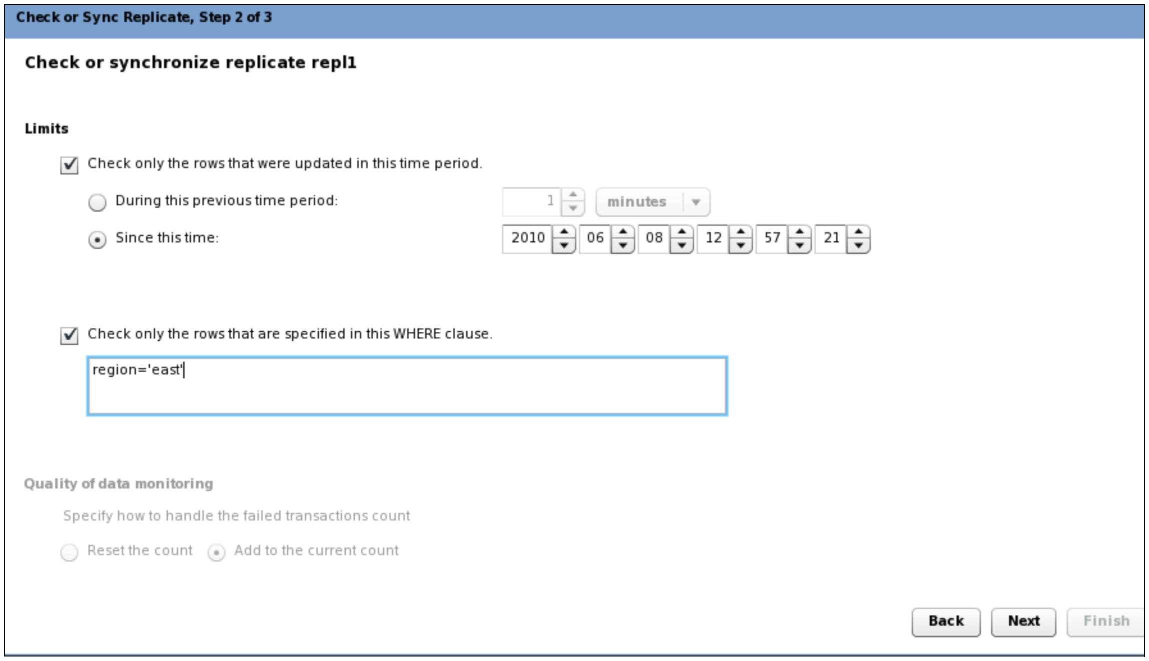
```
cdr check replicate r_employee .... --where="region='east'"
```

This command restricts the check of the employees to only those in the east region.

Using since and where with OAT

The since and where options to restrict the number of rows examined in a check replicate can be specified in OAT (Figure 5-74).

In this example, we are restricting the check to changes made since 12:57;21 on June 8, 2010. Also we are only including rows where the region is equal to “east”.


Check or Sync Replicate, Step 2 of 3

Check or synchronize replicate repl1

Limits

[✓] Check only the rows that were updated in this time period.

- During this previous time period: 1 minutes
- [● Since this time:] 2010 06 08 12 57 21

[✓] Check only the rows that are specified in this WHERE clause.

region='east'

Quality of data monitoring

Specify how to handle the failed transactions count

Reset the count [● Add to the current count]

Figure 5-74 Using OAT to perform a check with since and where restrictions

5.15.6 Using the ifx_replcheck column

If you have defined your table to include the ifx_replcheck columns, then the check of either a replicate or replicate set can be significantly reduced. The ifx_replcheck column is a hidden column that is created when the table is created WITH REPLCHECK, or altered by adding the REPLCHECK column.

Example 5-70 shows the statements to define or alter table to have a REPLCHECK column.

Example 5-70 Creating the ifx_replcheck column

```
create table mytab (...) with REPLCHECK;
alter table yourtab add REPLCHECK;
```

To be useful to the check, the ifx_replcheck column must be included in a composite index where the members of the index is <primary_key> + ifx_replcheck. The index must contain the primary key columns first followed by the eifx_replcheck column.

After this setting is in place, then the check can be performed by using a key-only scan of the index, which can reduce the time required to doing a check of 90%. There is nothing special that needs to be done to the `cdr check` command, as the tool will automatically detect the existence of the index and column on each of the nodes and will adjust its processing based on the existence of those columns.

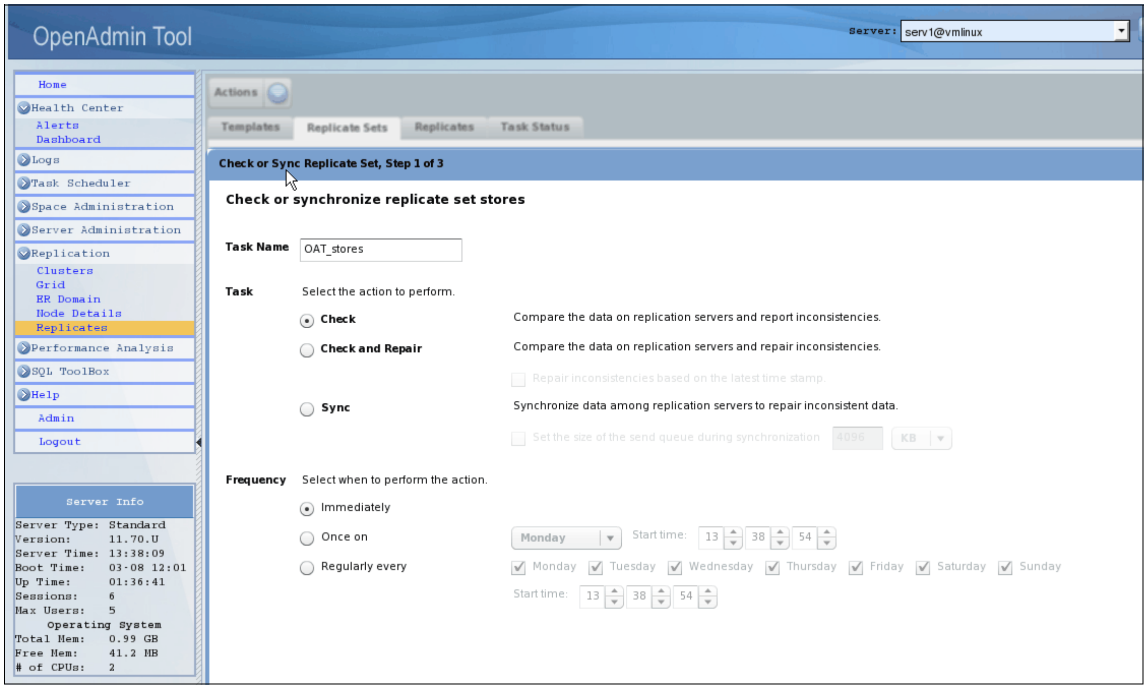
Note: The first check after creating the ifx_replcheck index may not be fast because the ifx_replcheck columns are not yet synchronized. Subsequent executions should be faster.

The advantage of using ifx_replcheck is that we only have to scan the index to determine if the rows are in sync. This is significantly faster than having to scan the data itself. However, this is not going to result in a reduction in the time to perform a one-time check. Its primary advantage is when you need to perform a check on a table periodically.

5.15.7 Scheduling a check using OAT

It is possible to schedule a check or synchronization by using OAT. This action posts a task in the sysadmin database that is responsible for performing the operation. This action is useful when it is desired to perform a check during off-peak times, or to perform periodic checks.

You can find the options to schedule a check or synchronization in the first window of the replicate or replicate set check/sync windows. You have seen this window several times in this chapter, so we do not explain how to get to it here. We prepare to perform a check on the stores_demo database and have reached the Check or Sync Replicate Set, Step 1 of 3 window, as shown in Figure 5-75.



Server Info

```
Server Type:  Standard
Version:      11.70.U
Server Time:  13:38:09
Boot Time:    03-08 12:01
Up Time:      01:36:41
Sessions:     6
Max Users:    5
     Operating System
Total Mem:    0.99 GB
Free Mem:     41.2 MB
# of CPUs:    2
```

Check or Sync Replicate Set, Step 1 of 3

Check or synchronize replicate set stores

Task Name OAT_stores

Task Select the action to perform.

[✓ **Check**] Compare the data on replication servers and report inconsistencies.

Check and Repair Compare the data on replication servers and repair inconsistencies.

Repair inconsistencies based on the latest time stamp.

Sync Synchronize data among replication servers to repair inconsistent data.

Set the size of the send queue during synchronization 4096 KB

Frequency Select when to perform the action.

[✓ Immediately]

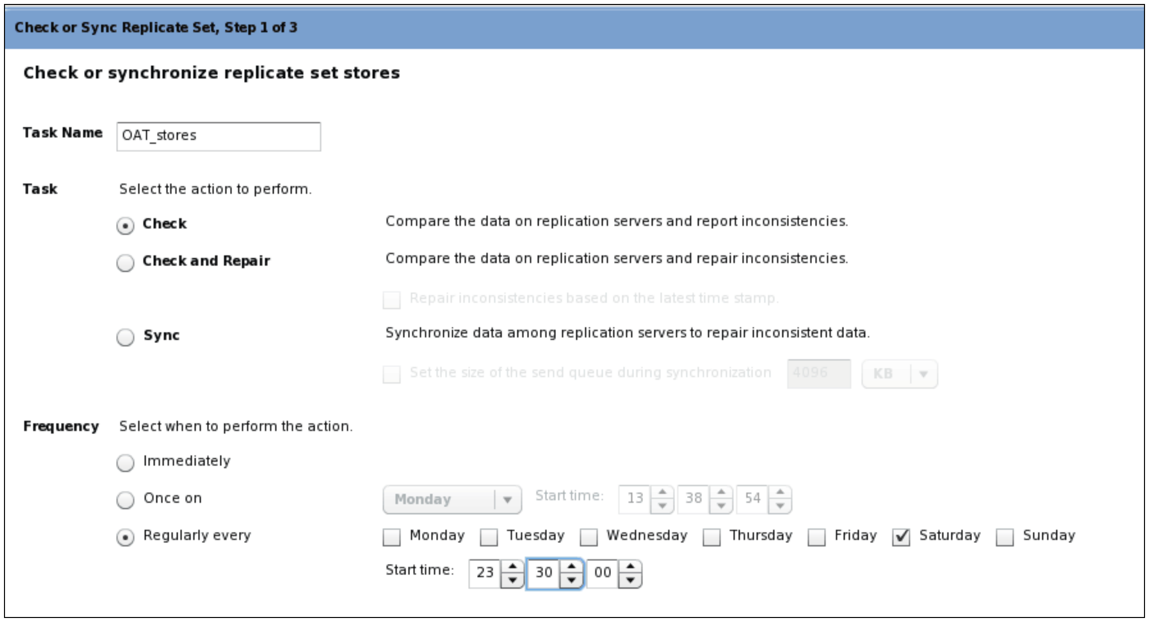
Once on Monday Start time: 13 38 54

Regularly every ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sunday

Start time: 13 38 54

Figure 5-75 Preparing for a check of a stores database

Up until now, the only thing that we have changed has been in the Task pane of the window. However, if we now want to perform an automatic check every Saturday at 11:30 p.m., we would change the settings to what are shown in Figure 5-76.


Check or Sync Replicate Set, Step 1 of 3

Check or synchronize replicate set stores

Task Name OAT_stores

Task Select the action to perform.

- [✓ **Check**] Compare the data on replication servers and report inconsistencies.
- **Check and Repair** Compare the data on replication servers and repair inconsistencies.
  - Repair inconsistencies based on the latest time stamp.
- **Sync** Synchronize data among replication servers to repair inconsistent data.
  - Set the size of the send queue during synchronization 4096 KB

Frequency Select when to perform the action.

- Immediately
- Once on Monday Start time: 13 38 54
- [✓ Regularly every] Monday Tuesday Wednesday Thursday Friday [✓ Saturday] Sunday
  Start time: 23 30 00

Figure 5-76 Scheduling a periodic check

This action starts a check of the database on each Saturday night. The results of the task are available in the Task Status tab.

We can also schedule a one time check or check repair by selecting **Once on** and specifying the day and start time of the task. This is useful if we are concerned about inconsistency and want to repair the inconsistency that night, but during off hours.

5.16 Updating configuration parameters

In Informix 11, you can dynamically update most of the Enterprise Replication configuration parameters. This action comes in handy if you need to update a configuration parameter while replication is active. For example, you realize that due to heavy concurrent workload on the target server, some of the replicated transactions, applied by data sync (DS) threads, are aborted as a result of lock contention between user threads and data sync threads. You can then increase the data sync lock wait time on that server by issuing **cdr change config** command, as shown in Example 5-71. You could verify the updated value using either **onstat**, a system catalog, or a server message log.

Example 5-71 Increase lock wait value for DS thread

- Update configuration parameter

```
$ cdr change config "CDR_DSLOCKWAIT 100"
```

- Verify update using onstat

```
$ onstat -g dss | grep LOCK
CDR_DSLOCKWAIT = 100
```

- Verify update using system catalog:

```
$ dbaccess sysmaster@nagpur -

Database selected.

> select * from sysconfig where cf_name="CDR_DSLOCKWAIT";

cf_id        152
cf_name      CDR_DSLOCKWAIT
cf_flags     0
cf_original  5
cf_effective 100
cf_default   5

1 row(s) retrieved.
```

- Verify update using message log

```
14:52:32  Value of CDR_DSLOCKWAIT has been changed to 100.
```

Note that the configuration parameter value is updated locally on the server where the command is executed. Changes to the configuration parameter are not replicated. However, you can perform an update on a remote server by using the --connect option of the **cdr** command.

The following commands are useful for making changes to the ER configuration parameters:

- **`cdr add config`**

  Appends the new values to the existing values.

- **`cdr change config`**

  Replaces the existing values with the new values.

- **`cdr remove config`**

  Removes the specified values from the existing values.

Usage of these commands shows you the variables that each command is allowed to modify.

5.17 Automating the administration

Informix provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server. This mechanism is called an *event alarm*. Events can be informative (for example, Backup Complete) or can indicate an error condition that requires your attention (for example, Unable to Allocate Memory). In this section, we describe how to automate administration of the Enterprise Replication servers using event alarms.

5.17.1 Use case

For example, over the weekend, your replication environment experienced an unplanned network outage. The target server becomes unavailable for replicating data changes. In this case, the source server starts to queue data changes in memory and eventually spools them to disk as memory fills. If the storage space allocated for the row data spooling also gets full, then the database server raises an event alarm to indicate that the spool storage is full. If the disk space is not added in time, then the replication could hang. In this situation, as a DBA, you need to log in to the system and add a new chunk to the spool storage.

What if you could automate the chunk addition whenever such an event occurs? With event alarms, you can automate the administrative tasks for such events.

5.17.2 Enabling event alarms

To automate the administration using event alarms, two things need to be done:

- Enable the events for which a database server should raise an alarm whenever such an event occurs.
- Set up the event alarm handler that would capture the event and perform the action you define.

By default, Enterprise Replication event alarms are enabled, except for the state change event alarms that are raised by administrative **cdr** commands. You can control which Enterprise Replication event alarms should be raised with the CDR_ENV configuration parameter.

To enable ER event alarms, complete the following steps:

1. Update the **onconfig** parameter CDR_ENV with the following setting:

   ```
   CDR_ENV CDR_ALARMS=30-39,47-50,71,73-75
   ```

2. Restart the database server.
3. Set the CDR_ALARMS to the list of event Class IDs for which you want the database server to raise an event alarm.

For the complete list of Enterprise Replication event alarms, go to the following address:

http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.erep.doc/ids_erp_286.htm

By default, Informix raises an event alarm only for the noteworthy events, such as events greater than severity 1. All Enterprise Replication event alarms are high severity, that is, greater than severity 1.

5.17.3 Customizing an alarm handler

After you enable the event alarms, you need to set up the alarm handler that captures the event and performs the action you define.

Informix provides a sample alarm handler script called `alarmprogram.sh` as part of Informix distribution; it is located in the `$INFORMIXDIR/etc` directory. Using this sample script, you can write your own shell script, batch file, or binary program that contains the event alarm parameters to handle the events.

Set the ALARMPROGRAM **`onconfig`** parameter to the full path name of this file. When an event occurs, the database server invokes this executable file and passes it the event alarm parameters shown in Table 5-4. For example, your script can use the _id and _msg parameters to take administrative action when an event occurs.

Table 5-4 Event alarm parameters

Parameter	Description	Data type
severity	The severity of the event.	integer
class_id	A numeric identifier that classifies the type of event that has occurred.	integer
class_msg	A brief messages that describes the classification of the event.	string
specific_msg	Specific messages that describes the event that occurred.	string
see_also	A reference to a file that contains additional information about the event.	string
uniqueid	A unique event identifier for the specific message.	bigint

Using the example from 5.17.1, "Use case" on page 243, how would you automate the chunk addition whenever Enterprise Replication spooled data storage runs out of space? When the database server encounters such an event, it raises an alarm with following event alarm parameters.

- `class_id = 31`
- `uniqueid = 31002`
- `class_msg = ER stable storage queue sbspace is full`
- `specific_msg = CDR QUEUER: Send Queue space is FULL - waiting for space in sbspace_name`

Clearly, the alarm handler program must be updated to capture the class_id = 31 event. The sample script provided by Informix does not handle class_id 31, so you need to add a new case statement. Because there are multiple event IDs for the same class, you have to perform a switch based on the event ID (that is, uniqueid). Add the code in this case statement that would add a chunk to the space configured by the CDR_QDATA_SBSPACE configuration parameter.

The next time the row data spool storage space becomes full, the alarm handler add a chunk automatically.

6 Monitoring and troubleshooting Enterprise Replication and Flexible Grid

Enterprise Replication in IBM Informix is one of the most powerful replication technologies available in any database server. It provides a highly flexible definition of the database server infrastructure to cover the predefined company's business requirements. ER enables a DBA to configure highly sophisticated and complex environments with various abilities, setting up the data flow based on heterogeneous environments in terms of database server versions and operating system types.

But complex environments also require a good understanding of the server components providing this functionality. Knowledge about the internal data flow, the threads, and, most importantly, the several monitoring facilities for early detection of problems, is quite necessary.

In addition to using the monitoring facilities, preparing to handle unexpected error situations is another requirement, which means looking into the appropriate diagnostic files and performing the necessary changes to solve problems. Being well prepared for that administration task enables the DBA running ER to meet business requirements, regardless of how sophisticated the topology.

6.1 How Enterprise Replication works internally

The previous chapters of this book have taken a detailed look into administration tasks to answer questions about how to set up and how to maintain an ER environment. This knowledge helps with understanding how replication servers and replicates are set up and maintained. But ER does not consist only of these two components. The replication server and replicates provide guidelines about how ER supports the internal server components.

But what are these components, how are they represented, and how do they work together? In this section, we give you an overview of these components.

In general, ER is designed to be updated anywhere by default, which means that each server has capture components and apply components (Figure 6-1).

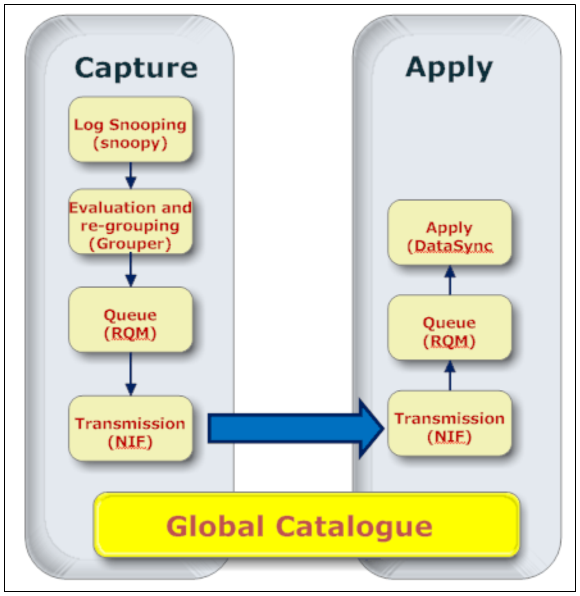


Figure 6-1 Basic design of Enterprise Replication

The capture consists of log snooping, evaluation of the log records, combining the log records into replicated transactions, placing the replicated transaction into a queue, and the transmission of the replicated transaction to the targets. The apply consists of receiving transactions that are transmitted from the source capture, the queuing of those transactions into a receive queue, and the apply of those transactions on the target server. The apply is supported by the global catalog that is found on all nodes and resides in the syscdr database. The global catalog contains all of the metadata that is used by Enterprise Replication. This metadata consists of what is being replicated from one server to another, information about the Flexible Grid, and progress tables, among other things.

ER uses several items of the Informix database in its implementation. It is important to understand how ER uses the following items:

- Memory:

  All local ER server components are in communication and exchanging data with the previous and next members in the transaction processing chain. This communication is performed by queues that are allocated in memory.

- Disk space:

  The following requests for disk space must be expected:

  – Send and receive queues when spooling
  – Syscdr database for system maintenance
  – Grouper page files for larger transactions

- Threads:

  ER defines many new threads in the system. The number of threads depends on the defined volume of replication servers that are linked together (CDRNr), the configuration parameter in the `onconfig` file (CDRGeval), and the current workload (DataSync).

  Either a thread or a certain group of threads represent a component and are part of the communication and workflow chain in the server.

Example 6-1 shows the threads that are automatically started when Enterprise Replication is started.

Example 6-1 ER related threads in the database server

```
36      1183a018 10d6ce58 1    sleeping secs: 16       1cpu        CDRSchedMgr
38      1183a838 10d6d9e8 1    sleeping secs: 260      1cpu        CDRDTCleaner
39      1183abc8 10d6dfb0 1    cond wait  CDRCparse    1cpu        CDRCparse
45      11c0ce08 10d6f108 3    sleeping secs: 1        1cpu        CDRGfan
46      11c99018 10d6f6d0 3    sleeping secs: 5        1cpu        CDRGeval0
47      11c993a8 10d6fc98 3    sleeping secs: 5        1cpu        CDRGeval1
48      11d54018 10d70260 3    sleeping secs: 5        1cpu        CDRGeval2
49      11d542f0 10d70828 3    sleeping secs: 5        1cpu        CDRGeval3
```

```
50      11bb2418 10d70df0 2    sleeping secs: 1        1cpu        ddr_snoopy
51      11bb2970 10d713b8 1    sleeping secs: 16       3cpu        CDRNsT2932
53      1203d2d8 10d71f48 1    sleeping secs: 2        1cpu       CDRRTBCleaner
54      1203d658 10d6bd00 1    cond wait  netnorm      3cpu        CDRNr2932
55      1203da88 10d72510 1    sleeping secs: 1        1cpu        CDRM_Monitor
56      1203dc78 10d72ad8 1    cond wait  CDRAckslp    1cpu        CDRACK_0
57      1203de68 10d730a0 1    cond wait  CDRAckslp    1cpu        CDRACK_1
58      11b28418 10d73668 1    cond wait  CDRDssleep   1cpu        CDRD_1
59      121fe068 10d73c30 1    cond wait  CDRDssleep   1cpu        CDRD_2
60      121fe2b8 10d71980 1    sleeping secs: 17       1cpu        CDRNsA2934
61      11bb2d68 10d741f8 1    cond wait  netnorm      1cpu        CDRNrA2934
```

To make understanding the general flow easier, first distinguish the ER components between the capture (source) and the apply (target):

- Global Catalog:
  - Contains a definition about the data flow and the database server.
  - Simplifies the sycdr database.
- Log snooper:
  - Responsible for reading the logical log files and extracting ER related log entries.
  - The related thread for this component is the ddr_snoopy thread.
- Grouper:
  - Attaches the extracted log entries provided by the ddr_snoopy thread to the appropriate open transaction maintained in internal lists.
  - After a transaction commit has been determined by the grouper, based on the replicate definitions taken from the global catalog (syscdr database), the transaction is evaluated for sending. This action includes:
    - Extracting only the changes covered by the definition of a replicate.
    - Combining changes on the primary key in the same transaction.
  - After the transaction is finally processed, it is prepared for the send to the several target servers defined by the replicate (fanout).
  - The threads attached to the grouper are CDRGeval[#] (the number depends on the `onconfig` file settings) and the CDRGfan.
- Queuer:

  Responsible for staging replicated transactions, control messages, and ACK sends until they are applied at all target replication nodes.

- Network interface (NIF):
  - Responsible for reliably transferring replicated transactions, and other metadata related to replication, using a TCP/IP interface.
  - The threads that ensure communication in the network and the determination of which members of the queue have to be sent out are the CDRN[rs][AT] threads.

For the final apply of the distributed transactions on the target side, the following components are defined:

- Network interface (NIF):
  - Responsible for reliably receiving replicated transactions, and other metadata related to replication, using a TCP/IP interface.
  - Similar threads on the send side are attached to this component.
- Receive Manager:

  Responsible for queuing replicated transactions to the receive queue and feeding the replicated transactions to data sync threads to enable re-applying of the transactions at target replication nodes.
- Data sync:
  - Responsible for re-applying replicated transactions at the replication target nodes.
  - One data sync thread handles only one transaction for the apply at one time, but multiple data sync threads can process multiple transactions in parallel even for operations on the same table.
  - In peak times, additional threads are automatically started for workload balancing.
  - There are the CDRD threads dedicated to apply the transactions on the target.

You may notice that not all the threads listed in Example 6-1 on page 249 are related to a subcomponent of ER. There are several additional threads for housekeeping, such as delete table cleanup, select definition parsing for new defined replicates, scheduling distribution times for replicates with frequencies defined, and monitoring threads.

6.2 How the Flexible Grid works internally

The Flexible Grid uses ER to perform data propagation and also to deliver the commands to the remote servers.

In the syscdr database, there is a set of tables that are automatically created when ER is initially started on a server. These tables include tables such as grid_cdr_node_tab, grid_def_tab, grid_part_tab, grid_cmd_tab, grid_cmd_part_tab, and so on. These tables are automatically replicated on all nodes within the ER domain.

When a grid operation is performed, the text of that operation is written to the grid_cmd_tab and grid_cmd_part_tab tables. Because these tables are replicated, then that means that the command is propagated to all of the nodes within the ER domain.

When these operations are applied on the target servers, Informix fires a trigger to record the grid operations to a work list. After coming out of the trigger, Informix spawns a thread that assumes the user identity and environment of the original user that executed the operation, checks to see if the operation should be performed on this server, and finally executes the operation on the target server.

If the execution is successful, then the grid operation is ACKed by writing a row in the grid_cmd_ack_tab table. If there was a problem, then a row is written into grid_cmd_errors_tab.

6.3 Monitoring Enterprise Replication

The different monitoring capabilities provided by Informix are strengths of the database server. The DBA can determine which interface is preferred and which granularity or amount of information needs investigating for the health or problem situations of the system. The following interfaces can be used for monitoring the different ER tasks in the system:

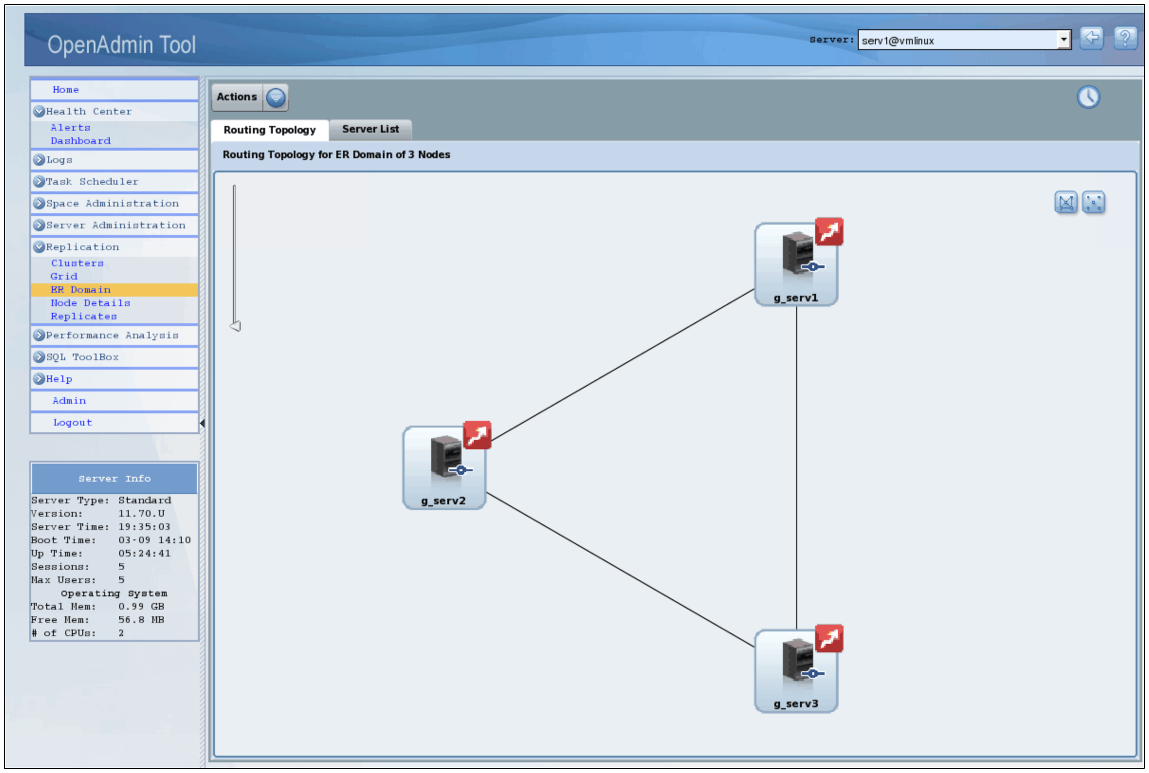
- OpenAdmin Tool (OAT):
  - OAT provides several tabs that are excellent for monitoring the ER system, both at a domain level and at the server level.
  - OAT also provides the ability to get alerts about a problem and provides drill-down support after a problem has been identified.

- The onstat utility:
  - onstat is the general tool that provides the DBA with the actual system data. The data is directly read from the shared memory and is always at the current level.
  - onstat provides a certain set of ER related options to monitor global catalog settings, the progress the execution of certain tasks, such as the snooper or the grouper, queue statistics, and network communications.
- The sysmaster database:
  - In addition to onstat, the sysmaster database provides an SQL based interface to read specific ER related information directly from memory.
  - Generating SQL statements for status investigation provides the DBA with the flexibility for remote administration and generating embedded web-based monitoring functionality bundled with defined alarms and alarm handling.
- The cdr utility:

  In general, the cdr utility is the workbench for cdr administration. It provides the interface to create, change, and drop all the necessary ER components, such as replication servers and replicates or templates. But in addition to the administration interface, there are a few monitoring options that can be used. In particular, replication server monitoring and replicate definition monitoring are available.

6.3.1 Using OAT to monitor Enterprise Replication

When you use OAT to monitor ER, you should first open the ER summary window by selecting **Replication** → **ER Domain**, as shown in Figure 6-2. Normally, the server icons do not have the red arrow box. However, if that server has sensed that something is wrong, then it will display the red arrow box as an alert to notify the DBA that investigation is required

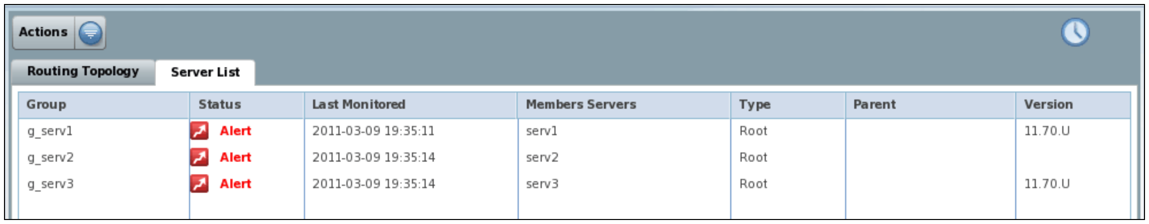


Routing Topology for ER Domain of 3 Nodes

```
Server Info
Server Type:  Standard
Version:      11.70.U
Server Time:  19:35:03
Boot Time:    03-09 14:10
Up Time:      05:24:41
Sessions:     5
Max Users:    5
     Operating System
Total Mem:    0.99 GB
Free Mem:     56.8 MB
# of CPUs:    2
```

Figure 6-2 ER domain page with alert notifications

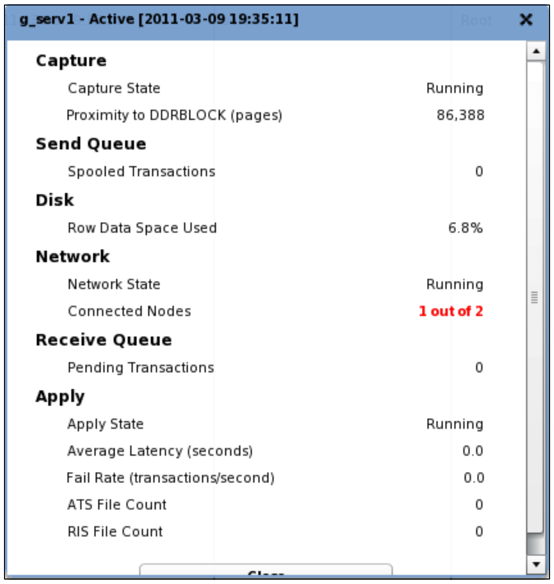
Figure 6-3 shows the Server List tab.



Group	Status	Last Monitored	Members Servers	Type	Parent	Version
g_serv1	Alert	2011-03-09 19:35:11	serv1	Root		11.70.U
g_serv2	Alert	2011-03-09 19:35:14	serv2	Root		
g_serv3	Alert	2011-03-09 19:35:14	serv3	Root		11.70.U

Figure 6-3 Server List summary

To learn more detailed information about possible problems, click the alert icon (). For example, if we click the alert button for serv1, we see the details shown in Figure 6-4.

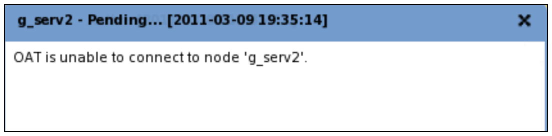


g_serv1 - Active [2011-03-09 19:35:11]

Capture	
Capture State	Running
Proximity to DDRBLOCK (pages)	86,388
Send Queue	
Spooled Transactions	0
Disk	
Row Data Space Used	6.8%
Network	
Network State	Running
Connected Nodes	1 out of 2
Receive Queue	
Pending Transactions	0
Apply	
Apply State	Running
Average Latency (seconds)	0.0
Fail Rate (transactions/second)	0.0
ATS File Count	0
RIS File Count	0

Figure 6-4 Detailed server information for serv1

We learn that we are not able to connect to one of the nodes. If we click the alert button for serv2, we discover the problem (Figure 6-5).



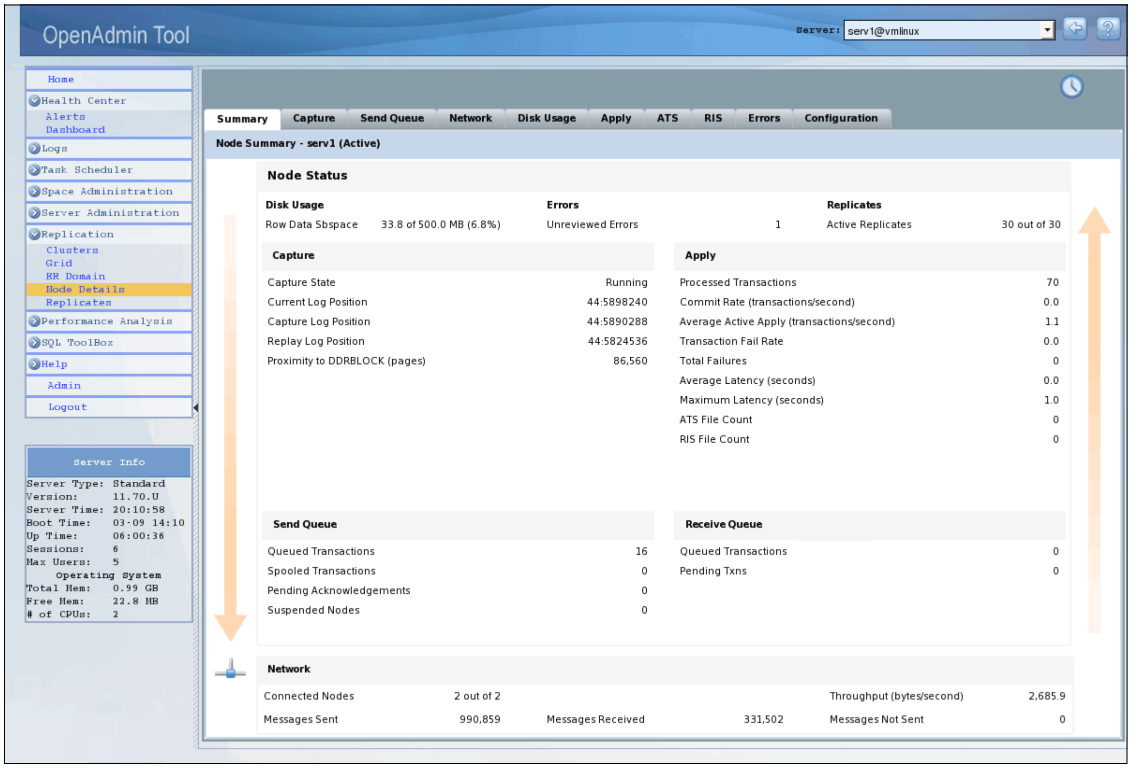
g_serv2 - Pending... [2011-03-09 19:35:14]

OAT is unable to connect to node 'g_serv2'.

Figure 6-5 Detailed information about serv2

We learn that serv2 is not running, so we simply restart serv2.

Examining the detail windows can help identify potential problem spots as well. To perform this task, we make serv1 the current server by using the servers drop-down menu and navigating to the Node Details window by selecting **Replication** → **Node Details** (Figure 6-6).



Server Info

```
Server Type:  Standard
Version:      11.70.U
Server Time:  20:10:58
Boot Time:    03-09 14:10
Up Time:      06:00:36
Sessions:     6
Max Users:    5
     Operating System
Total Mem:    0.99 GB
Free Mem:     22.8 MB
# of CPUs:    2
```

Node Summary - serv1 (Active)

Node Status

Disk Usage		Errors		Replicates	
Row Data Sbspace	33.8 of 500.0 MB (6.8%)	Unreviewed Errors	1	Active Replicates	30 out of 30

Capture	
Capture State	Running
Current Log Position	44:5898240
Capture Log Position	44:5890288
Replay Log Position	44:5824536
Proximity to DDRBLOCK (pages)	86,560

Apply	
Processed Transactions	70
Commit Rate (transactions/second)	0.0
Average Active Apply (transactions/second)	1.1
Transaction Fail Rate	0.0
Total Failures	0
Average Latency (seconds)	0.0
Maximum Latency (seconds)	1.0
ATS File Count	0
RIS File Count	0

Send Queue	
Queued Transactions	16
Spooled Transactions	0
Pending Acknowledgements	0
Suspended Nodes	0

Receive Queue	
Queued Transactions	0
Pending Txns	0

Network					
Connected Nodes	2 out of 2			Throughput (bytes/second)	2,685.9
Messages Sent	990,859	Messages Received	331,502	Messages Not Sent	0

Figure 6-6   ER Node Summary tab

The Node Summary window displays an overall view of a single server. The flow of data is indicated by the orange arrows on the sides of the window with the left side representing the data collection on the source side and the right side representing the target apply. Because the default behavior of Enterprise Replication is update anywhere, each node within the ER domain is both a source and a target. The bottom of the window contains network information, because the network is the connector between the source capture and the target apply. In effect, this window follows the basic design of Enterprise Replication.

By default, each of these windows is refreshed with new information after a minute. You can change the refresh rate by clicking the clock at the upper right part of the window and adjusting the slide bar (Figure 6-7).

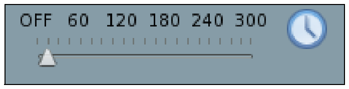
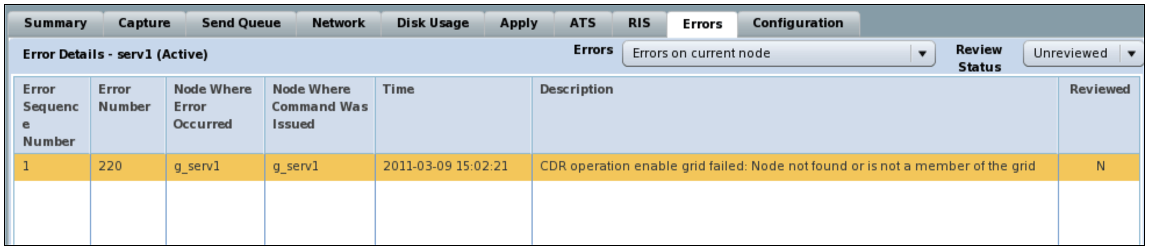


Figure 6-7 Adjusting the refresh time bar

In our example, we notice that there is one unreviewed error. We want to review this error, so we open the Errors tab (Figure 6-8).



Error Details - serv1 (Active)

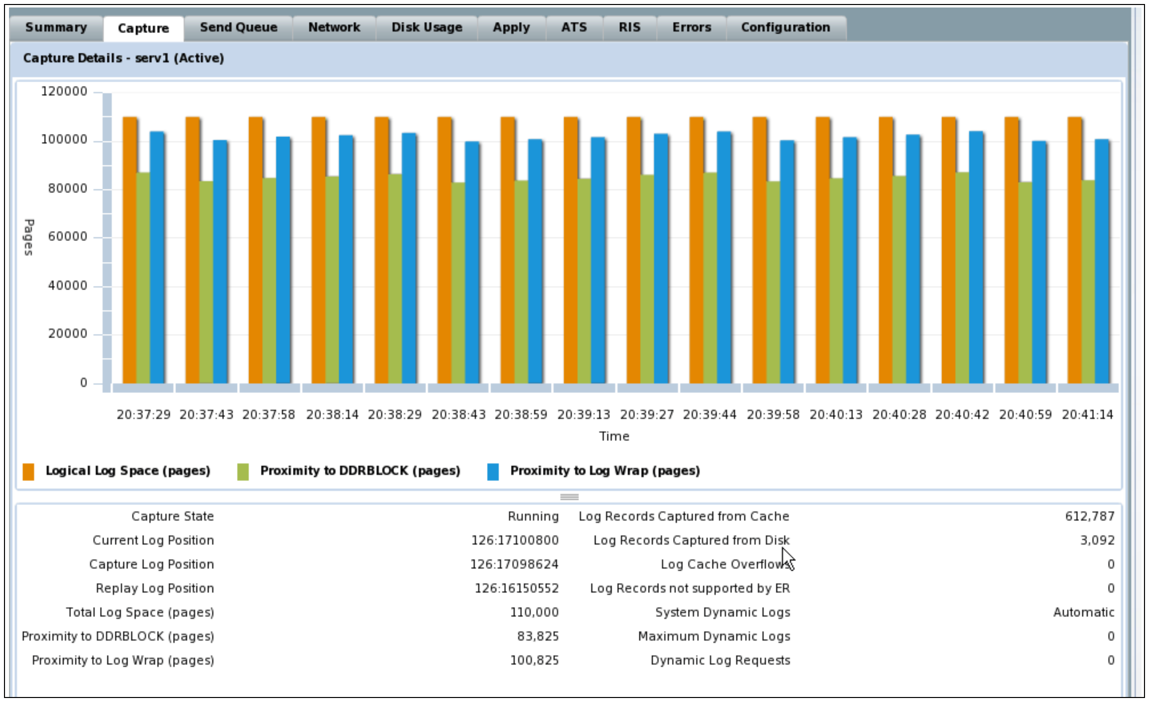
Errors: Errors on current node

Review Status: Unreviewed

Error Sequence Number	Error Number	Node Where Error Occurred	Node Where Command Was Issued	Time	Description	Reviewed
1	220	g_serv1	g_serv1	2011-03-09 15:02:21	CDR operation enable grid failed: Node not found or is not a member of the grid	N

Figure 6-8 Errors tab

Although this is an error, it does not look like a serious problem. Perhaps it is nothing but a typo, so we check open the Capture tab (Figure 6-9).


Summary
Capture
Send Queue
Network
Disk Usage
Apply
ATS
RIS
Errors
Configuration
Capture Details - serv1 (Active)
Pages
Time
Logical Log Space (pages)
Proximity to DDRBLOCK (pages)
Proximity to Log Wrap (pages)

Capture State	Running	Log Records Captured from Cache	612,787
Current Log Position	126:17100800	Log Records Captured from Disk	3,092
Capture Log Position	126:17098624	Log Cache Overflows	0
Replay Log Position	126:16150552	Log Records not supported by ER	0
Total Log Space (pages)	110,000	System Dynamic Logs	Automatic
Proximity to DDRBLOCK (pages)	83,825	Maximum Dynamic Logs	0
Proximity to Log Wrap (pages)	100,825	Dynamic Log Requests	0

Figure 6-9 Capture tab

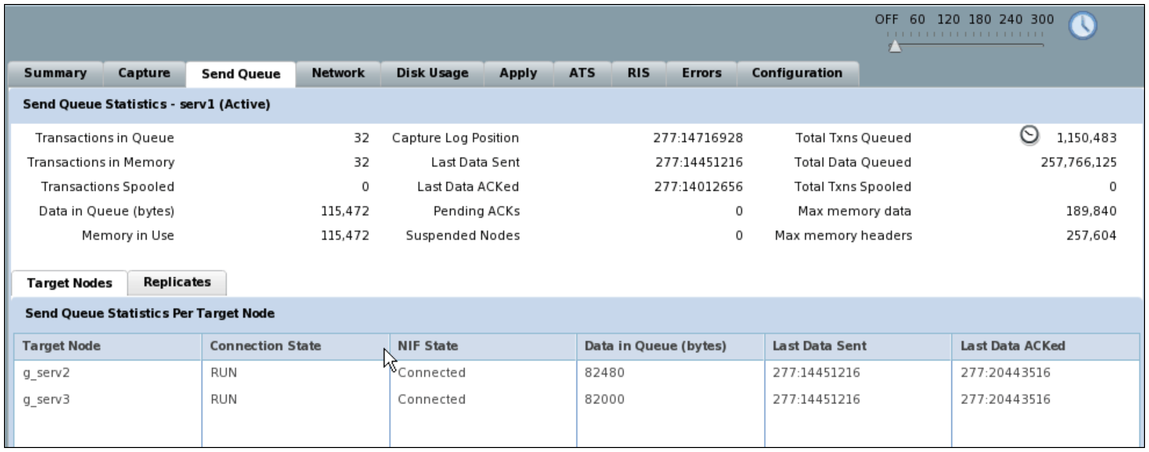
The key items to monitor in the capture window is a potential log wrap or a DDRBLOCK state. This items can be seen graphically by observing the green bar (DDRBLOCK) or blue bar (log wrap). Basically, the lower the bar, the greater the risk of a problem.

DDRBLOCK is a state that ER can get into while trying to prevent from a log wrap. This state blocks normal user activity in an attempt to give more resources to the spooling of transactions, which also decreases the probability of a log wrap.

When a log wrap occurs, there is a danger of running into a problem when the server is restarted. The “Replay Log Position” is the point in the logs from which ER will start recovery. There is a processing impact in having to spool a replicated transaction so the server normally avoids spooling replicated transactions and instead re-snoops the logs to rebuild transactions that need to be replicated. However, if the server is about to run into a log wrap, then the server starts spooling to disk so that it can recover the transactions.

Note: Consider using dynamic logs or log spooling to avoid the impact of a DDRBLOCK state. This way, ER adds a log file, or stages the logs to an external directory rather than going into a DDRBLOCK state.

The Send Queue tab shows information about what is in the queue to be transmitted to target nodes. The initial window displays what information is queued for each of the target nodes (Figure 6-10). The top half of the window identifies information about the send queue in general. The first two columns ("Transactions in Queue" through "Suspended Nodes") are a display of current information about the queue, while the third column ("Total Txns Queued" through "Max memory headers") displays information since ER was last started.



Send Queue Statistics - serv1 (Active)

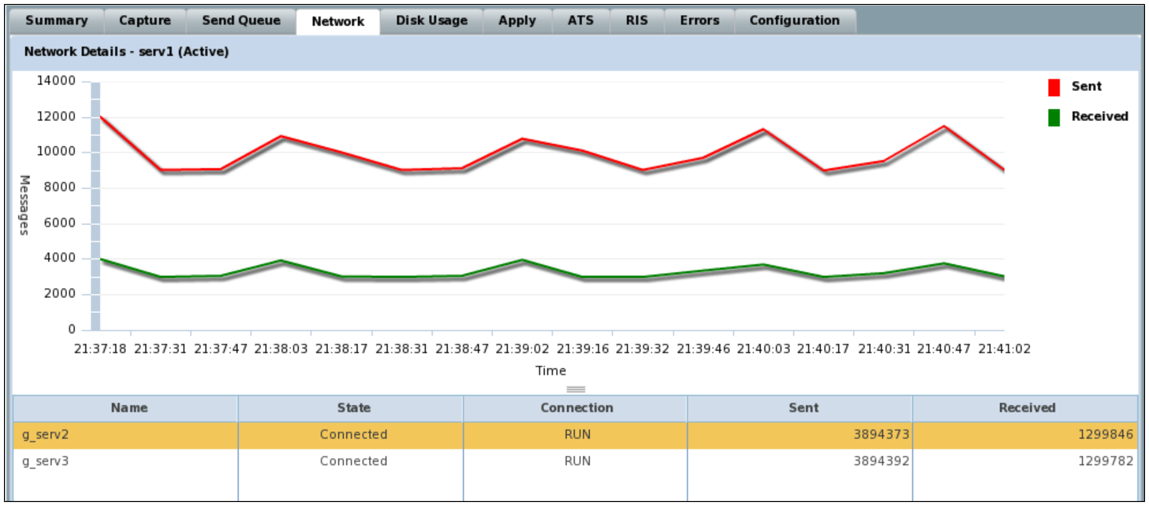
Transactions in Queue	32	Capture Log Position	277:14716928	Total Txns Queued	1,150,483
Transactions in Memory	32	Last Data Sent	277:14451216	Total Data Queued	257,766,125
Transactions Spooled	0	Last Data ACKed	277:14012656	Total Txns Spooled	0
Data in Queue (bytes)	115,472	Pending ACKs	0	Max memory data	189,840
Memory in Use	115,472	Suspended Nodes	0	Max memory headers	257,604

Send Queue Statistics Per Target Node

Target Node	Connection State	NIF State	Data in Queue (bytes)	Last Data Sent	Last Data ACKed
g_serv2	RUN	Connected	82480	277:14451216	277:20443516
g_serv3	RUN	Connected	82000	277:14451216	277:20443516

Figure 6-10 Send Queue tab

The Network tab shows information about the network. The top part shows a history of the messages that have been sent and received on this server. The bottom half displays statistics by each peer server (Figure 6-11).



Name	State	Connection	Sent	Received
g_serv2	Connected	RUN	3894373	1299846
g_serv3	Connected	RUN	3894392	1299782

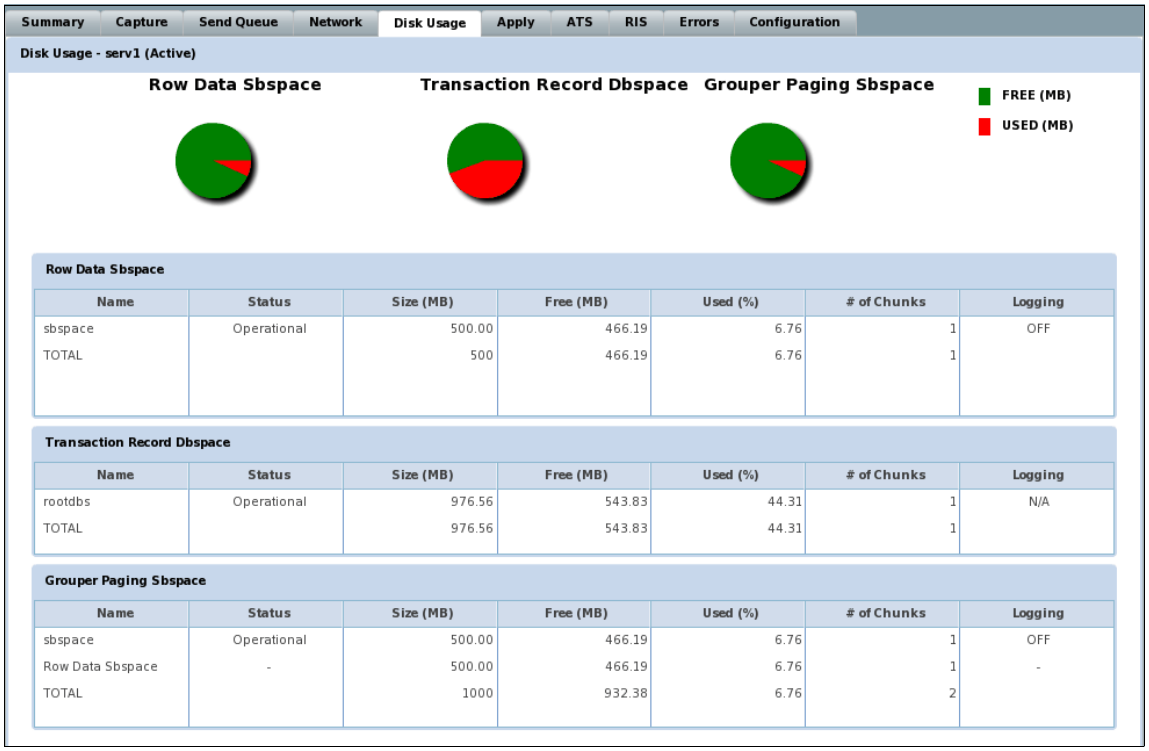
Figure 6-11 Network tab

The Disk Usage window (Figure 6-12) displays information about the disk space used by ER. This window is useful for identifying when space might be getting tight. Using this information, you can expand those spaces by adding a new chunk before running into trouble.

Row Data Sbspace is the smart BLOB space used to store transactions in a stable fashion. There can be multiple smartblob spaces used to store transactions. The graphic displays a summary of all of the spaces. If the space becomes full, any future transactions cannot be spooled, and that could cause a blockage of the system.

The Transaction Record Dbspace is where we store transaction headers. The size of the transaction header is much smaller than the transaction data, so the space utilization for the transaction record dbspace is not as heavily consumed as the row data sbspace.

The Grouper Paging Sbspace is used to contain large transactions that are in the process of being evaluated for replication.



Row Data Sbspace

Name	Status	Size (MB)	Free (MB)	Used (%)	# of Chunks	Logging
sbspace	Operational	500.00	466.19	6.76	1	OFF
TOTAL		500	466.19	6.76	1	

Transaction Record Dbspace

Name	Status	Size (MB)	Free (MB)	Used (%)	# of Chunks	Logging
rootdbs	Operational	976.56	543.83	44.31	1	N/A
TOTAL		976.56	543.83	44.31	1	

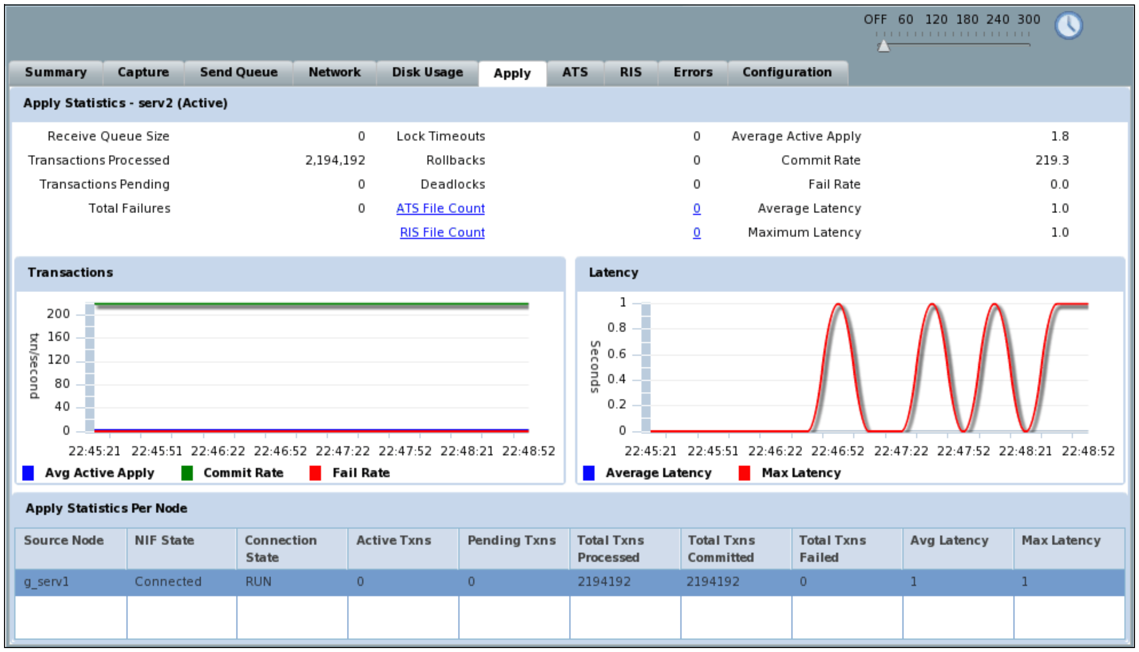
Grouper Paging Sbspace

Name	Status	Size (MB)	Free (MB)	Used (%)	# of Chunks	Logging
sbspace	Operational	500.00	466.19	6.76	1	OFF
Row Data Sbspace	-	500.00	466.19	6.76	1	-
TOTAL		1000	932.38	6.76	2	

Figure 6-12 Disk Usage tab

The Apply tab (Figure 6-13) displays information about the target apply. The target apply window identifies certain key information, such as how rows are being applied. The top pane shows key information about the apply performance, such as the degree of parallelism that is being used, the average commit rate in terms of commits per second, failure in the apply, and latency rates. Generally, the smaller the sizes of transactions, the greater the degree of parallelism and the lower the latency.

Note: A best practice to improve performance is to use multiple smaller transactions than a single large transaction.



Apply Statistics - serv2 (Active)

Receive Queue Size	0	Lock Timeouts	0	Average Active Apply	1.8
Transactions Processed	2,194,192	Rollbacks	0	Commit Rate	219.3
Transactions Pending	0	Deadlocks	0	Fail Rate	0.0
Total Failures	0	ATS File Count	0	Average Latency	1.0
		RIS File Count	0	Maximum Latency	1.0

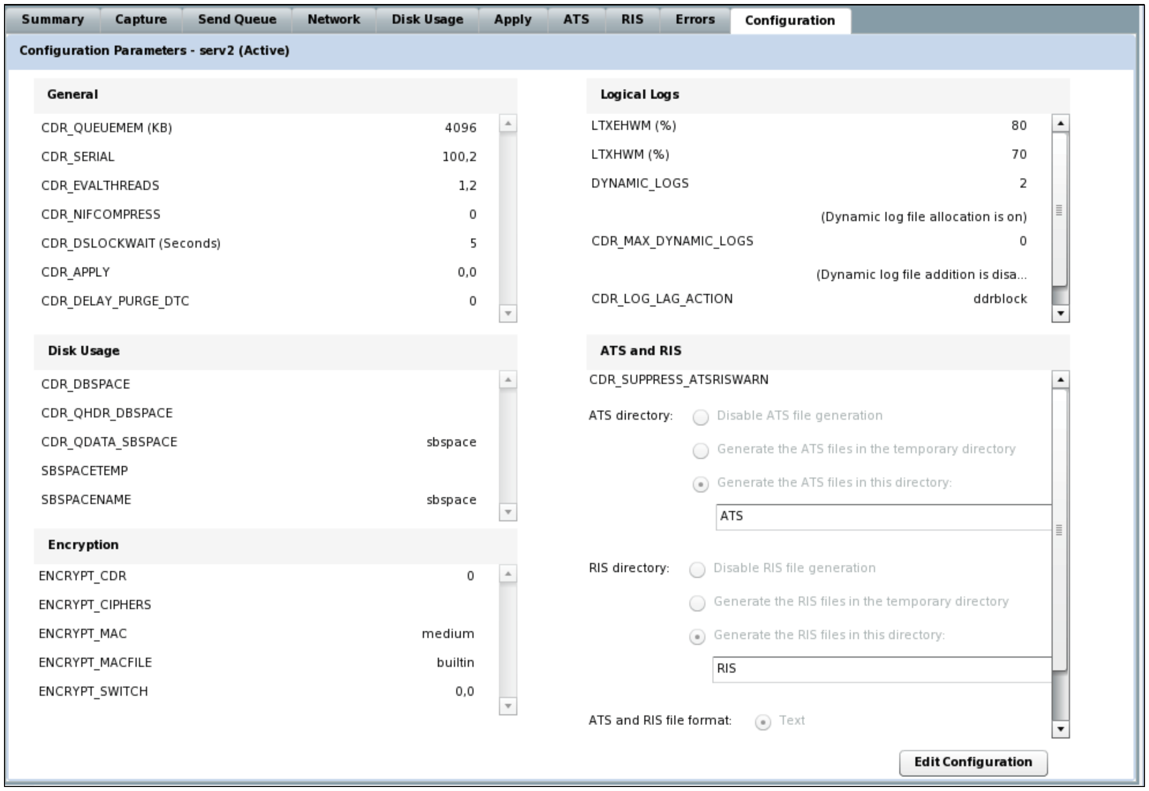
Apply Statistics Per Node

Source Node	NIF State	Connection State	Active Txns	Pending Txns	Total Txns Processed	Total Txns Committed	Total Txns Failed	Avg Latency	Max Latency
g_serv1	Connected	RUN	0	0	2194192	2194192	0	1	1

Figure 6-13 Apply tab

For information about the usage of the ATS and RIS windows, refer to Chapter 5, “Administering Enterprise Replication” on page 101.

The last tab is the Configuration tab (Figure 6-14).



Configuration Parameters - serv2 (Active)

General

CDR_QUEUEMEM (KB)	4096
CDR_SERIAL	100,2
CDR_EVALTHREADS	1,2
CDR_NIFCOMPRESS	0
CDR_DSLOCKWAIT (Seconds)	5
CDR_APPLY	0,0
CDR_DELAY_PURGE_DTC	0

Disk Usage

CDR_DBSPACE	
CDR_QHDR_DBSPACE	
CDR_QDATA_SBSPACE	sbspace
SBSPACETEMP	
SBSPACENAME	sbspace

Encryption

ENCRYPT_CDR	0
ENCRYPT_CIPHERS	
ENCRYPT_MAC	medium
ENCRYPT_MACFILE	builtin
ENCRYPT_SWITCH	0,0

Logical Logs

LTXEHWM (%)	80
LTXHWM (%)	70
DYNAMIC_LOGS	2
	(Dynamic log file allocation is on)
CDR_MAX_DYNAMIC_LOGS	0
	(Dynamic log file addition is disa...
CDR_LOG_LAG_ACTION	ddrblock

ATS and RIS

CDR_SUPPRESS_ATSRISWARN

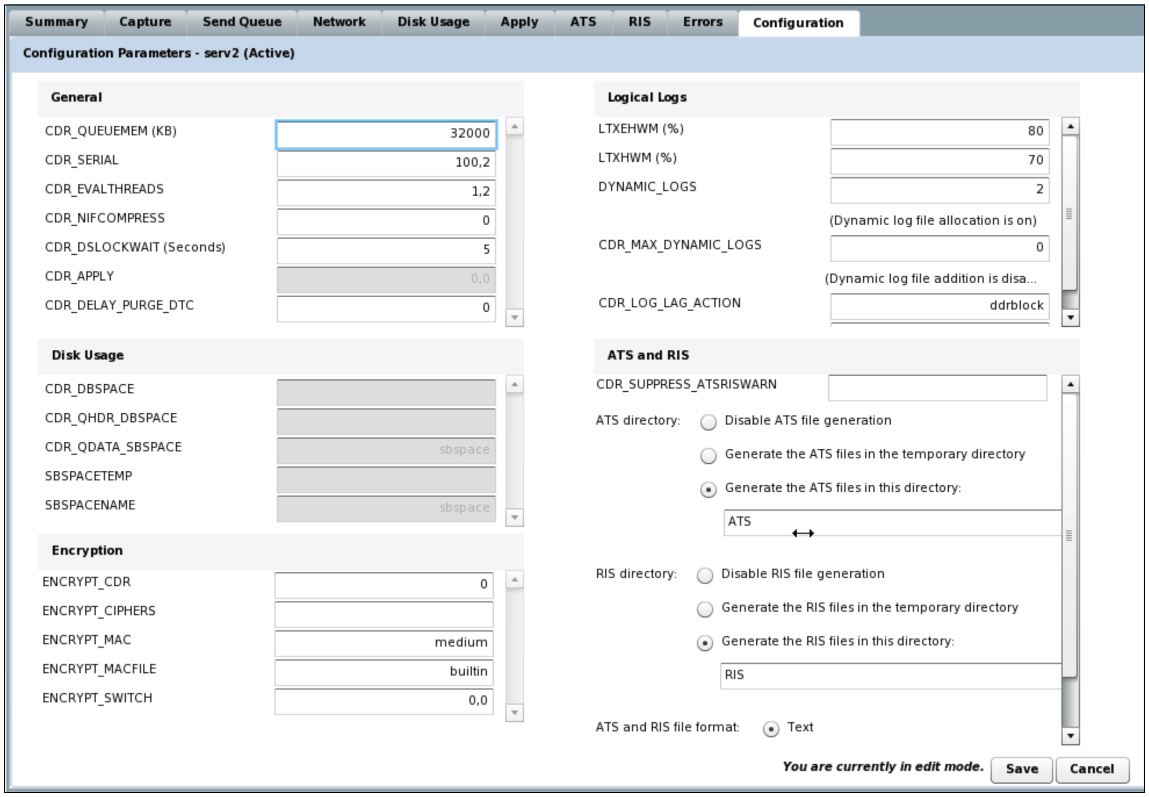
ATS directory: Disable ATS file generation / Generate the ATS files in the temporary directory / [✓ Generate the ATS files in this directory:] ATS

RIS directory: Disable RIS file generation / Generate the RIS files in the temporary directory / [✓ Generate the RIS files in this directory:] RIS

ATS and RIS file format: [✓ Text]

Figure 6-14 Configuration tab

This window displays the current configuration settings for ER, and can also be used to dynamically change those settings. For example, if we want to change the queue memory size, we can simply click **Edit Configuration**, enter the new value for CDR_QUEUEMEM, and then click **Save** to activate the change (Figure 6-15).



Configuration Parameters - serv2 (Active)

General

CDR_QUEUEMEM (KB)	32000
CDR_SERIAL	100,2
CDR_EVALTHREADS	1,2
CDR_NIFCOMPRESS	0
CDR_DSLOCKWAIT (Seconds)	5
CDR_APPLY	0,0
CDR_DELAY_PURGE_DTC	0

Disk Usage

CDR_DBSPACE	
CDR_QHDR_DBSPACE	
CDR_QDATA_SBSPACE	sbspace
SBSPACETEMP	
SBSPACENAME	sbspace

Encryption

ENCRYPT_CDR	0
ENCRYPT_CIPHERS	
ENCRYPT_MAC	medium
ENCRYPT_MACFILE	builtin
ENCRYPT_SWITCH	0,0

Logical Logs

LTXEHWM (%)	80
LTXHWM (%)	70
DYNAMIC_LOGS	2
	(Dynamic log file allocation is on)
CDR_MAX_DYNAMIC_LOGS	0
	(Dynamic log file addition is disa...
CDR_LOG_LAG_ACTION	ddrblock

ATS and RIS

CDR_SUPPRESS_ATSRISWARN

ATS directory: Disable ATS file generation / Generate the ATS files in the temporary directory / [✓ Generate the ATS files in this directory:] ATS

RIS directory: Disable RIS file generation / Generate the RIS files in the temporary directory / [✓ Generate the RIS files in this directory:] RIS

ATS and RIS file format: [✓ Text]

You are currently in edit mode.

Figure 6-15 Performing a dynamic change to a configuration item

After changing the configuration item in OAT, the `onconfig` file for that server is be updated as well. Example 6-2 shows the updated `onconfig` file.

Example 6-2 Updated the onconfig file

```
CDR_EVALTHREADS 1,2
CDR_DSLOCKWAIT  5
CDR_QUEUEMEM    32000
CDR_NIFCOMPRESS 0
CDR_SERIAL      100,2
CDR_DBSPACE
```

Note: Updating the configuration might require dynamically restarting portions of ER. For example, if the encryption rules are dynamically changed, then the network threads have to be restarted. Therefore, it is best not to change the encryption rules while processing large transactions.

6.3.2 Onstat and ER components

In addition to using OAT, you can also use onstat to display information about memory structures of the running Informix instance. This is especially useful when working with technical support because the text output of onstat is much easier to email to a support engineer than a window output.

First, you have to consider the overall architecture of ER. To obtain information about the ER main components, use different **`onstat`** monitoring options for each of the components. Figure 6-16 illustrates the relationship between components and **`onstat`** commands.

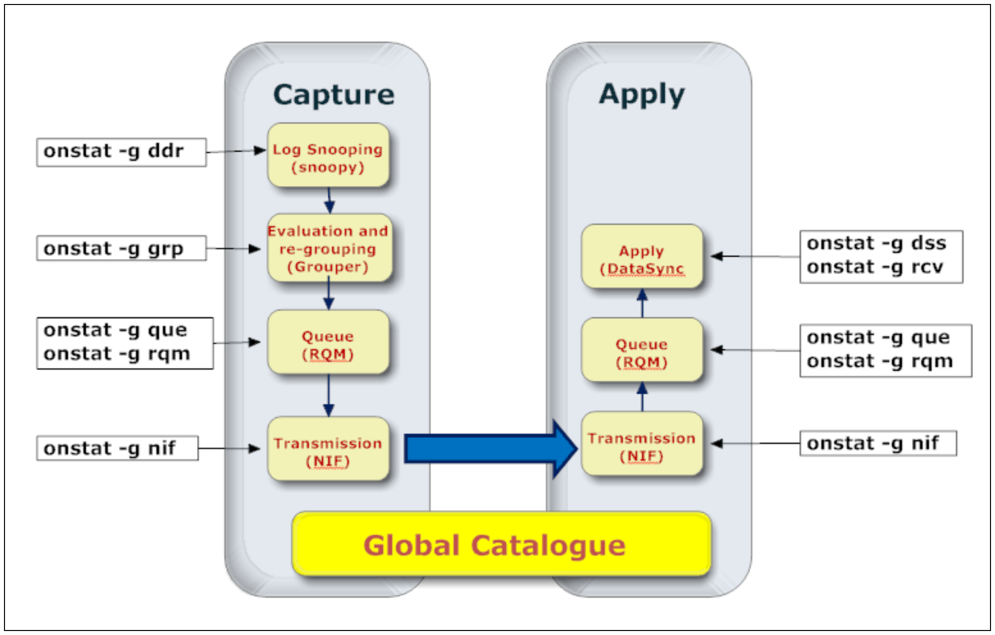


Figure 6-16 Relationship between components and onstat commands

6.3.3 Global catalog statistics

In an ER environment, the DBA has to distinguish between two things. There are some base elements, such as replication servers and replicates, that are defined by the DBA from outside of the database server by using the cdr utility. These definitions determine how the internal ER components in the database server, such as the snooper, grouper, queuer, data sync, or NIF, have to behave.

When obtaining information about the replication server, the replicates, and the replicate sets, all of the introduced monitoring interfaces return similar information.

Monitoring the replication server

We have, in a sample ER infrastructure, a hierarchical replication server definition. newyork is the root, frankfurt is defined as non-root, and singapore and london are defined as a leaf nodes. The appropriate output, taken from the output of the **`onstat -g cat`** command, is shown in Example 6-3.

Example 6-3 onstat -g cat servers

```
$onstat -g cat servers

GLOBAL-CATALOG CACHE STATISTICS
-------------------
Current server  : Id 2930, Nm newyork
  Last server slot: (0, 4)
  # free slots    : 0
  Broadcast map   : <[000d]>
  Leaf server map : <[0014]>
  Root server map : <[0002]>
  Adjacent server map: <[000c]>
    Id: 2930, Nm: newyork, Or: 0x0002, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 00, ishub: TRUE, isleaf: FALSE
      subtree map: <[001c]>
        NifInfo: 117f1238
                Last Updated    (0) 1970/01/01 01:00:00
                State           Local server
                Total sent      0 (0 bytes)
                Total recv'd    0 (0 bytes)
                Retry           0 (0 attempts)
                Connected       0

    Id: 2932, Nm: singapore, Or: 0x0004, off: 0, idle: 0, state Active
      root Id: 2930, forward Id: 2932, ishub: FALSE, isleaf: TRUE
        NifInfo: 117f12c0
                Last Updated    (1185846605) 2007/07/31 03:50:05
                State           Connected
                Total sent      3 (24 bytes)
                Total recv'd    1 (60 bytes)
                Retry           0 (0 attempts)
                Connected       1
    Id: 2934, Nm: frankfurt, Or: 0x0008, off: 0, idle: 0, state Active
      root Id: 2930, forward Id: 2934, ishub: TRUE, isleaf: FALSE
      subtree map: <[0010]>
        NifInfo: 117f1360
                Last Updated    (1185846606) 2007/07/31 03:50:06
                State           Connected
                Total sent      3 (24 bytes)
                Total recv'd    1 (60 bytes)
                Retry           0 (0 attempts)
                Connected       1

    Id: 2936, Nm: london, Or: 0x0010, off: 0, idle: 0, state Active
      root Id: 2934, forward Id: 2934, ishub: FALSE, isleaf: TRUE
        NifInfo: 117f13e8
                Last Updated    (0) 1970/01/01 01:00:00
                State           Never connected
                Total sent      0 (0 bytes)
                Total recv'd    0 (0 bytes)
                Retry           0 (0 attempts)
                Connected       0
```

In Example 6-3 on page 266, the following conventions are used:

- Id: Server group ID.
- Nm: Group name.
- Or: OR bit. Unique bit for each server.

  The OR bit is useful in understanding output from many other **`onstat`** commands. For example, it can be used to discover which servers are yet to acknowledge a transaction in the send queue (**`onstat -g rqm sendq`**).
- NeedAck: Waiting for Acks from `<[0004]>`.
- state: Shows server status.

  Valid states are `Quiescent`, `Active`, and `Suspended`. If a state is shown as `Quiescent`, then there is a problem with the server definition.
- isleaf: If set to `TRUE`, it is a leaf server.

Similar to an **`onstat -g cat`** output, a **`cdr list server`** command returns information in a more verbose style. The different options that can be used for the call are shown in Example 6-4.

Example 6-4 cdr list server

```
#general output
$cdr list server
SERVER                ID STATE    STATUS     QUEUE  CONNECTION CHANGED
-----------------------------------------------------------------------
frankfurt           2934 Active   Connected       0 Jul 31 03:50:06
london              2936 Active                   0
```

```
newyork             2930 Active   Local           0
singapore           2932 Active   Connected       0 Jul 31 03:50:05

#Output for a Root server
$cdr list server frankfurt
NAME                ID     ATTRIBUTES
---------------------------------------
frankfurt           2934 root=newyork hub

#Outut for a Leaf server
$cdr list server london
NAME                ID     ATTRIBUTES
---------------------------------------
london              2936 root=frankfurt forward=frankfurt leaf
```

The sysmaster interface provides the ability to select the data from the syscdrserver or syscdrs tables. A sample output from both tables for one of the defined replication servers is shown in Example 6-5.

Example 6-5 Select from syscdrs and syscdrserver sysmaster tables

```
$echo "select * from syscdrs " | dbaccess -e sysmaster


servid         2930
servname       newyork
cnnstate       L
cnnstatechg    0
servstate      A
ishub          Y
isleaf         N
rootserverid   0
forwardnodeid  0
timeout        0


$ echo "select * from syscdrserver " | dbaccess -e sysmaster

servid           2932
servername       singapore
connstate        C
connstatechange  1185846605
servstate        Active
ishub            N
isleaf           Y
rootserverid     2930
forwardnodeid    2932
idletimeout      0
```

```
atsdir          /vobs/IDS11/holgerk/ATS
risdir          /vobs/IDS11/holgerk/RIS
```

Monitoring the replicates

To generate a short output, there is only one replicate defined in our sample ER environment. This replicate defines an update-anywhere scenario with stored procedure conflict resolution. See the output of the **`onstat -g cat repls`** command for a detailed description of all defined attributes for the replicate, as shown in Example 6-6.

Example 6-6 onstat -g cat repls

```
$onstat -g cat repls

GLOBAL-CATALOG CACHE STATISTICS

REPLICATES
-------------------
Parsed statements:
          Id 192020553 table customer

  Inuse databases: stores9(1)
    Name: raadr1, Id: 192020553 State: ACTIVE Flags: 0   use 0 lastexec Thu Jan  1 01:00:00 1970

        Local Participant: stores9:informix.customer
        Attributes: TXN scope, Enable ATS, all columns sent in updates
        Conflict resolution[Prim::Sec]: [STOREDPROCEDURE conflict_acct_sp ]
        Delete shadow table: cdr_deltab_000040
        Column Mapping: ON, columns INORDER, offset 8, uncomp_len 134
        No Replicated UDT Columns

        Bitmap: all <[001e]> route <[000c]>
```

The following conventions are used in Example 6-6:

- Name: Name of the replicate provided during replicate definition or system generated (when a template is realized).
- Id: Replicate ID, which is internally generated by the server. It is unique in the ER network of servers, and is generated using two numbers:
  - Server ID in SQLHOSTS
  - A serial number per Informix server
- State: Current state of the replicate. Valid states are `FAILED`, `SUSPENDED`, `INACTIVE`, `ACTIVE`, `DELETED`, `QUIESCENT`, and `RESUMED`. If you see replicate state set to `FAILED`, the replicate definition is in an inconsistent state.

Typically, the cdr utility is used for the definition and maintenance of the ER elements. In addition to that functionality, there is an option to list the defined replicates known by the replication server.

The output for the **cdr list replicate** call is shown in Example 6-7.

Example 6-7 cdr list replicate output

```
$ cdr list repl

CURRENTLY DEFINED REPLICATES
-------------------------------
REPLICATE:        raadr1
STATE:            Active ON:newyork
CONFLICT:           conflict_acct_sp
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.customer
OPTIONS:          transaction,ats,fullrow
REPLID:           192020553 / 0xb720049
REPLMODE:         PRIMARY  ON:newyork
APPLY-AS:         INFORMIX ON:newyork

$ cdr list repl brief raadr1

REPLICATE           TABLE                                   SELECT
----------------------------------------------------------------------------
raadr1 stores9@newyork:informix.customer      select * from informix.customer
raadr1 stores9@singapore:informix.customer    select * from informix.customer
raadr1 stores9@frankfurt:informix.customer    select * from informix.customer
raadr1 stores9@london:informix.customer       select * from informix.customer
```

Run a SELECT for the syscdrrepl in the sysmaster database to get the appropriate output for replicates definitions. A example is shown in Example 6-8.

*Example 6-8 Select * from syscdrrepl*

```
$ echo "select * from syscdrrepl " | dbaccess -e sysmaster

Database selected.

replname          raadr1
replstate         Active
replid            192020553
freqtype          C
freqmin
freqhour
freqday
scope             T
invokerowspool    N
invoketranspool   Y
primresolution    S
secresolution
```

```
storedprocname    conflict_acct_sp
ismasterrepl      N
isshadowrepl      N
shadowparentrepl
floattype         N
istriggerfire     N
isfullrow         Y
```

6.3.4 Monitoring the global ER variable settings

Informix 11 provides functionality to change most of the ER related parameter settings without restarting the server. The cdr utility provides the interface to achieve that task. This functionality includes the possibility to monitor the actual settings. The **onstat** option -g cdr config is provided by Informix 11. Example 6-9 shows the output using this option.

Example 6-9 onstat -g cdr config

```
$onstat -g cdr config
IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 02:56:57 -- 51904 Kbytes
CDR_DBSPACE:
    CDR_DBSPACE configuration setting:                          syscdr
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: cdr_sbspace
CDR_QHDR_DBSPACE:
    CDR_QHDR_DBSPACE configuration setting:                          rootdbs
CDR_QUEUEMEM:
    CDR_QUEUEMEM configuration setting:       4096
CDR_SERIAL:
    CDR_SERIAL configuration setting:          1, 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]
```

An additional possibility for getting the CDR specific **onconfig** parameters is provided in the sysconfig table in the sysmaster database. This table contains the settings for all **onconfig** parameters. Because the CDR related parameters typically start with the "CDR" prefix, the selection can be done with matches, as shown in Example 6-10.

Example 6-10 Select the actual settings for the onconfig parameter from sysmaster

```
select * from sysconfig where cf_name matches "CDR*"

cf_id         152
cf_name       CDR_DSLOCKWAIT
cf_flags      0
cf_original   5
cf_effective  5
cf_default    5
....
```

6.4 Monitoring the source replication server

Now that we have described the monitoring facilities for general components and settings, in this section we look at the ER subcomponents. The discussion about the tasks for the source and target replication servers is divided into two sections. The discussion of source components starts from the read of the log entries and finishes at the distribution over the network to the target. The status and statistics interface of the components is introduced in the order of the transaction processing chain.

6.4.1 Log snooping statistics

Enterprise Replication uses log-based data capture to gather data for replication, reading the logical log to obtain the row images for tables that participate in replication.

Informix database servers manage the logical log in a circular fashion, where 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, the user transactions are blocked until Enterprise Replication can advance. This situation is called DDRBLOCK mode; it occurs only if the system is severely misconfigured.

Part 3, "Logging and log administration", of the *IBM Informix Administrator's Guide*, SC27-3526, describes how to configure the logical logs for Enterprise Replication. It is difficult to reach the best configuration on the first attempt because it depends on many factors that are only known over time as the implementation progresses. To understand the logical log requirements and monitor the system, onstat provides the log snooping statistics option.

The ddr_snoopy thread is the first thread in the ER transaction process chain on the source side. The assigned task reads sequentially through the log files, generated by the daily workload in the system. In addition, all logical log entries that are related to ER replicate definitions are captured and queued for the grouper.

The **onstat -g ddr** command can be used for monitoring the progress and the statistics for that particular thread, as shown in Example 6-11.

Example 6-11 onstat -g ddr

```
$onstat -g ddr

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:10:52 -- 51904 Kbytes

DDR -- Running --

# Event  Snoopy   Snoopy   Replay   Replay   Current  Current
Buffers   ID      Position  ID      Position   ID     Position
1040     102      315018   102      1731fc   102      316000

Log Pages Snooped:
      From      From      Tossed
     Cache      Disk  (LBC full)
        20       770           0

Total dynamic log requests: 0
```

```
DDR events queue

Type   TX id    Partnum  Row id
```

Typically, when the logical logs are configured correctly and ER is operating properly, you should see the message `DDR -- Running --`, as shown in Example 6-11 on page 273. If there is a problem in the system due to misconfiguration of the log, you may see the message `Blocked:DDR` in the status line, as shown in Example 6-12.

Example 6-12 DDR blocked state

```
IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 03:02:46 -- 3 6864 Kbytes
Blocked:DDR
```

The value for DDRBLOCK is set when the current log position is getting too close to the replay log position. Example 6-12 shows the **onstat** output when the server is in DDRBLOCK mode.

In Informix servers at Version 9.3x or later, you should rarely see this situation. If you do see this situation, then make sure that you have configured the logical log files properly. For example, you should not see frequent logical log switches. Ideally, you should make sure that there is a logical log switch in 60 minute intervals.

Another possibility might be that the send queue smart blob space (CDR_QDATA_SBSPACE configuration parameter) is full and the replication threads are waiting for more space in the smartblob space to enable them to spool replication data to disk.

Here is a list of details regarding certain fields from the **onstat -g ddr** command's output:

- Replay position: This is the oldest transaction that has not been acknowledged by all of its target instances and has not been stably stored in the send queue. The replay position is the point where ER would have to begin snooping if the engine is restarted.
- Snoopy position: The snoopy position is the log position that ER (ddr_snoopy) is currently processing.
- Current position: The current log position is where the sqlexec threads are currently adding records to the logical log.
- Total dynamic log requests: Total number of dynamic log requests made by ER. If the CDR_MAX_DYNAMIC_LOGS **onconfig** parameter is enabled, then ER may request dynamic log addition to avoid a DDRBLOCK situation.

The supported values for the CDR_MAX_DYNAMIC_LOGS **onconfig** parameter are:

- 0: Disable dynamic log addition in case of a DDRBLOCK scenario.
- Positive integer: Limit the total number of dynamic log requests to this number.
- -1: Always request dynamic log addition in DDRBLOCK scenarios. This setting is not recommended, as it may consume all your disk space if ER is waiting for additional disk space in the send queue smart blob space while spooling send queue data to disk.

Note: To enable ER dynamic log addition functionality, the DYNAMIC_LOGS **onconfig** parameter value must be set to 2.

6.4.2 Grouper statistics

The next stage in the chain on the source side is the grouper. The grouper is a subcomponent in ER that receives transactions from the snooper and evaluates the log records for replication. This action is done based on the replicate definition. The grouper evaluates the queued log records and attaches them to the appropriate open transaction. After a commit for one of the current processed transactions is seen in a log entry, the grouper rebuilds the transaction by compressing operations on the same primary key to one activity within the CDRGeval thread and queues the transaction for transmission with the CDRGfan thread. The DBA can configure multiple grouper threads in the server, which can work in parallel.

The **onstat -g grp** command can be used to see the grouper statistics, as shown in Example 6-13.

Example 6-13 onstat -g grp

```
$onstat -g grp

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:17:40 -- 51904 Kbytes
Grouper at 0x11c15018:
Last Idle Time: (1185847628) 2007/07/31 04:07:08
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
Conflict Resolution Blocks Allocated: 0
```

```
Memory pool cache: Empty
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 0
Tx returned from Queuer: 0
Events sent to Queuer: 0
Events returned from Queuer: 0
Total rows sent to Queuer: 0
Open Tx array size: 1024
Complete log ID, position: 102,1520124
Table 'customer' at 0x11d021b0 [ CDRShadow ]
```

Here are some especially useful information elements from Example 6-13 on page 275:

- Open Tx: Logs are sent to the grouper as soon as they are generated by an active transaction. This number represents the open transactions for which the grouper has not yet received a “commit work” or “rollback work” log record.
- Serial Tx: The grouper received all log records for these transactions.
- Tx not sent: The number of transactions that were rejected by the grouper. These transactions have been discarded primarily because they were rolled back.
- Tx sent to Queuer: Number of transactions that the grouper queued for replication.
- Tx returned from Queuer: These transactions are either stably stored on disk or applied at all target servers.

Grouper replicate statistics

The grouper statistics also produce information about replicates. For example, the grouper-related statistics can be generated by the **onstat -g grp R** command, as shown in Example 6-14.

Example 6-14 onstat -g grp R

```
$onstat -g grp R

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:18:50 -- 51904 Kbytes
Replication Group 192020553 at 0x11dc8d08
  Replication at 0x11dc6260 192020553:192020553 (customer) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs IeeeFloats InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 134
```

If **onstat -g cat repls** shows that the replicate state is `Active`, then you should not see `Ignore` or `Stopped` in the grouper replicate statistics.

Note: This is one of the places to look if you observe that data is not getting replicated for one or more replicated tables.

Grouper paging statistics

The grouper statistics also produce paging statistics. The grouper works on active transactions and stores them locally until the transaction completes (commit or rollback). It can run out of memory if the transaction happens to be long. It is recommended that you avoid long transactions whenever possible. Refer to the section “Setting Up the Grouper Paging File” in the *Informix Enterprise Replication Guide*, SC27-3539.

The **`onstat -g grp pager`** command provides the current status of the paging system by the grouper, as shown in Example 6-15.

Example 6-15 onstat -g grp pager

```
$onstat -g grp pager

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:24:13 -- 51904 Kbytes
Grouper Pager statistics:
Number of active big transactions: 1
Total number of big transactions processed: 0
Spool size of the biggest transaction processed: 0 Bytes
```

The output in Example 6-15 shows large transaction statistics. If the system is experiencing replication latency, make sure that replication of large transactions is not causing the delay. Run **`onstat -g grp pager`** at the source server and make sure that the large transaction count is zero.

6.4.3 Reliable Queue Manager (RQM) statistics on the source

After the grouper thread finishes the packaging of the current transaction for distribution, the transaction is sent to the send queue. In addition to the send queue, the RQM maintains several other queues dedicated to tasks, such as maintaining control messages (for example, a replicate definition has changed) or maintaining ACKs (handshaking) on the source side.

Depending on the network throughput, the availability of the target, the replicate definition for the replication, or the workload on the target, the content of the queue can remain in memory if necessary. If the memory is exhausted, especially for the send queue, the content of the queue has to be spooled to disk. This causes the utilization of disk space in a normal dbspace for header information and a sblobspace for the packaged transaction itself.

It is necessary to monitor the queues on both sides to determine performance bottlenecks and malfunctions, because if the defined dbspaces are running out of space, ER cannot continue.

The **onstat** command provides detailed information about the queues. The command used for viewing queue activity is **onstat -g rqm**. This command is further divided with a number of suboptions, such as sendq and cntrlq.

Send Queue statistics

The **onstat** command produces detailed output as part of the sendq statistics. Here we have a sample output from **onstat -g rqm sendq** divided into two parts (shown in Example 6-16 and Example 6-17 on page 279).

Example 6-16 onstat -g rqm sendq (part 1)

```
$ onstat -g rqm sendq

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 02:45:30 -- 51904 Kbytes

CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0x11dcd018) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 89/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:             59
 Log Events in queue:       2
 Txns in memory:            25
 Txns in spool only:        34
 Txns spooled:              50
 Unspooled bytes:           741402
 Size of Data in queue:     6777360 Bytes
 Real memory in use:        847296 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 1060072 (12595200) Bytes
 Max Real memory hdrs used  640440 (12595200) Bytes
 Total data queued:         7839448 Bytes
 Total Txns queued:         89
 Total Txns spooled:        51
 Total Txns restored:       49
 Total Txns recovered:      0
 Spool Rows read:           33243
 Total Txns deleted:        30
 Total Txns duplicated:     0
 Total Txn Lookups:         271

 Progress Table:
       Progress Table is Stable
               On-disk table name............:         spttrg_send
               Flush interval (time).........:         30
               Time of last flush............:         1185856506
               Flush interval (serial number):         1000
```

```
            Serial number of last flush...:          9
            Current serial number.........:          9
```

Some of the important output elements from the send queue statistics are:

- Txns in queue: Total number of transactions in the queue.
- Txns in memory: Total number of transactions that are in memory. These transactions may also be spooled to disk.
- Txns in spool only: Total number of transactions on disk only. These transactions are removed from memory.
- Txns spooled: Total number of transactions on disk. Some of these transactions may also be in memory.
- Pending Txn Buffers: Number of buffers queued in the send queue for a partial transaction. Grouper is currently queuing these transaction buffers into the send queue and the transaction is not yet completely queued.

Historical data

Send queue statistics also report historical data. This historical data is useful for understanding the load and performance of the system. Looking at the historical data can help when trying to configure the system for optimal performance. Here are a few examples of the historical data:

- Max real memory data used and Max real memory hdrs used: This is the maximum real memory used at any time. The value in the brackets is the high water mark.
- Total data queued: Total number of bytes queued at this time.
- Total Txns queued: Total number of transactions queued at this time.
- Total Txns spooled: Total number of transactions spooled at this time.

Example 6-17 shows part 2 of the detailed output produced by **onstat** as part of the sendq statistics.

Example 6-17 onstat -g rqm sendq (part 2)

```
Server    Group Bytes Queued     Acked                    Sent
-----------------------------------------------------------------------------
  2934 0xb72004e 4731264 efffffff/efffffff/efffffff/efffffff - b72/ab/2152e8/0
  2932 0xb72004e 443556  b72/aa/27b648/0         - b72/ab/2152e8/0
  2934 0xb720049 0       b72/66/16d154/0 -efffffff/efffffff/efffffff/efffffff
  2932 0xb720049 0       b72/66/1731fc/0 -efffffff/efffffff/efffffff/efffffff

 First Txn (0x1227ff48) Key:  2930/166/0x0012a5b8/0x00000000
 Txn Stamp: 58/0, Reference Count: 0.
 Txn Flags: Spooled, Restored
 Txn Commit Time: (1185856453) 2007/07/31 06:34:13
 Txn Size in Queue: 105894
 First Buf's (0x123e8df0) Queue Flags: None
 First Buf's Buffer Flags: None
 NeedAck: Waiting for Acks from <[0008]>
 No open handles on txn.

 Last Txn (0x128bd5b8) Key:  2932/38/0x00000017/0x00000000
 Txn Stamp: 89/0, Reference Count: 0.
 Txn Flags: None
 Txn Commit Time: (1185856507) 2007/07/31 06:35:07
 Txn Size in Queue: 72
 First Buf's (0x128bd670) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Event, Event_Mode
 NeedAck: Waiting for Acks from <[0008]>
 No open handles on txn.

 Traverse handle (0x11dde230) for thread CDRNsT2932 at txn (0x1249bc00)
End_of_Q,  Flags: None
 Traverse handle (0x11df0230) for thread CDRNr2932 at Head_of_Q,  Flags: None
 Traverse handle (0x12482110) for thread CDRGeval0 at Head_of_Q,  Flags: None
 Traverse handle (0x12482c50) for thread CDRACK_0 at Head_of_Q,  Flags: None
 Traverse handle (0x12483630) for thread CDRACK_1 at Head_of_Q,  Flags: None
 Traverse handle (0x12484010) for thread CDRGeval1 at Head_of_Q,  Flags: None
 Traverse handle (0x123e8018) for thread CDRGeval3 at Head_of_Q,  Flags: None
 Traverse handle (0x12334018) for thread CDRGeval2 at Head_of_Q,  Flags: None
```

Progress table information

Example 6-18 shows the format the progress table information.

Example 6-18 Progress table format

```
Server      Group  Bytes  Queued      Acked          Sent
-----------------------------------------------------------------
```

In a progress table dump, the fields in the table are:

- Server: Target server ID.
- Group: Replicate ID.
- Bytes: Bytes queued for this replicate ID from this server ID.
- Acked: Last transaction key acknowledged from this server.

- Sent: Last txn queued in the send queue for this server.

  If either the last send point or last ack point is unknown, either of these transaction keys can be seen in the progress table:

  ```
  0xfeeeeeee/0xfeeeeeee/0xfeeeeeee/0xfeeeeeee
  0xefffffff/0xefffffff/0xefffffff/0xefffffff
  ```

- Transaction key: This is a four part key made up of the following items:
  - Source server ID
  - Logical log ID of the transaction commit position in the logical log file
  - Logical log position of the transaction commit position in the logical log
  - Sequence number
- NeedAck: Waiting for Acks from <[0008]>: The NeedAck bitmap shows the bit values of the servers from which this transaction is waiting for the acknowledgement. To get the bits that belong to server, use the **onstat -g cat servers** command's output. Look for the `Or` from this output, as shown in the following excerpt from Example 6-3 on page 266:

  ```
  Id: 2930, Nm: newyork, Or: 0x0002, off: 0, idle: 0, state Active
        root Id: 00, forward Id: 00, ishub: TRUE, isleaf: FALSE
        subtree map: <[001c]>
      Id: 2932, Nm: singapore, Or: 0x0004, off: 0, idle: 0, state Active
        root Id: 2930, forward Id: 2932, ishub: FALSE, isleaf: TRUE
      Id: 2934, Nm: frankfurt, Or: 0x0008, off: 0, idle: 0, state Active
        root Id: 2930, forward Id: 2934, ishub: TRUE, isleaf: FALSE
        subtree map: <[0010]>
      Id: 2936, Nm: london, Or: 0x0010, off: 0, idle: 0, state Active
        root Id: 2934, forward Id: 2934, ishub: FALSE, isleaf: TRUE
  ```

A similar output for each rqm queue provides the sysmaster interface. There is a syscdr_rqm table in the sysmaster database. A SELECT of the rows from that particular table shows columns that have the same meaning when compared with the **onstat** output. In Example 6-19, only the output for the send queue is shown, but there are additional rows for the other queues as well.

Example 6-19 Select the queue statistics from the sysmaster table

```
$echo "select * from syscdr_rqm" | dbaccess -e sysmaster

rqm_idx           0
rqm_name          trg_send
rqm_flags         137473
rqm_txn           2039
rqm_event         1
rqm_txn_in_memory 387
rqm_txn_in_spool_+ 1652
rqm_txn_spooled   1765
```

```
rqm_unspooled_byt+  144762
rqm_data_in_queue   1080212
rqm_inuse_mem       204652
rqm_pending_buffer  0
rqm_pending_data    0
rqm_maxmemdata      205712
rqm_maxmemhdr       230256
rqm_totqueued       10847396
rqm_tottxn          21173
rqm_totspooled      2953
rqm_totrestored     814
rqm_totrecovered    0
rqm_totspoolread    2199
rqm_totdeleted      19146
rqm_totduplicated   13
rqm_totlookup       119199
```

Due to the importance of monitoring, it is good to have a closer look into the sysmaster and the syscdr interface in an attempt to get more detailed information about the send queue.

The sysmaster interface provides, for most of the tables in the RQM, two tables for selecting status information for all transaction headers stored in the send queue. They are syscdrsend_txn and syscdrsend_buf. If there are transactions in the system to be sent out, as is the case with the **`onstat -g rqm sendq`** command's output, only the numbers can be seen. In addition, the sysmaster provides information for each transaction stored in the queue. The output looks much like that shown in Example 6-20.

Example 6-20 sysmaster information for the sendq

```
$ echo "select * from syscdrsend_txn " | dbaccess -e sysmaster

ctkeyserverid  2930
ctkeyid        172
ctkeypos       3756572
ctkeysequence  0
ctstamp1       100
ctstamp2       0
ctcommittime   1185856509
ctuserid       0
ctfromid       0

....
```

Control queue statistics

The difference between the control queue and other queues is that data in the control queue is always spooled to disk. CDR Admin commands, such as **`cdr start replicate`** and **`cdr stop replicate`**, are replicated through this control queue.

Example 6-21 shows a sample output of the control queue statistics generated by **`onstat -g rqm cntrlq`**.

Example 6-21 onstat -g rqm cntrlq

```
RQM Statistics for Queue (0x11dd6018) control_send
 Transaction Spool Name: control_send_stxn
 Insert Stamp: 386/0
 Flags: CTRL_SEND_Q, STABLE, USERTXN, PROGRESS_TABLE, NEED_ACK
 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: 3938 (0) Bytes
 Max Real memory hdrs used  1320 (0) Bytes
 Total data queued:        7132 Bytes
 Total Txns queued:        8
 Total Txns spooled:       8
 Total Txns restored:      0
 Total Txns recovered:     0
 Spool Rows read:          0
 Total Txns deleted:       8
 Total Txns duplicated:    0
 Total Txn Lookups:        51

 Progress Table:
        Progress Table is Stable
                On-disk table name............:         sptcontrol_send
                Flush interval (time).........:         30
                Time of last flush............:         1185848005
                Flush interval (serial number):         1000
                Serial number of last flush...:         2
                Current serial number.........:         4

Server    Group Bytes Queued     Acked                     Sent
------------------------------------------------------------------------------
```

```
  2934      0x1         0 b72/0/199/0              -            b72/0/199/0
  2932 0xb740000        0 b72/0/198/0              -            b72/0/198/0
```

The control queue is used for replicating and transferring replication definitions. With replication objects (such as replicate, replicate set, and servers), the status differs from server to server, so this is one of the places to look to find pending control messages.

Example 6-22 shows sample output of the control queue when a **delete replicate** command could not be sent to the target server. The first few lines, which are similar to Example 6-21 on page 283, are omitted here. You can see that 153 bytes of control data is queued for server 200, and it also waiting for an acknowledgement.

Example 6-22 onstat -g rqm cntrlq

```
Server    Group Bytes Queued      Acked                    Sent
------------------------------------------------------------------------------
  2934      0x1      153 b72/0/199/0            -           b72/0/19b/0
  2932 0xb740000       0 b72/0/19a/0            -           b72/0/19a/0

 First Txn (0x12482018) Key:  2930/0/0x0000019b/0x00000000
 Txn Stamp: 388/0, Reference Count: 0.
 Txn Flags: Spooled
 Txn Commit Time: (1185848783) 2007/07/31 04:26:23
 Txn Size in Queue: 185
 First Buf's (0x123e89f8) Queue Flags: Spooled, Resident
 First Buf's Buffer Flags: Control
 NeedAck: Waiting for Acks from <[0009]>
 Traverse handle (0x11df9230) for thread CDRNsT2932 at txn (0x12482018) End_of_Q,  Flags: None
 Last Txn (0x12482018) Key:  2930/0/0x0000019b/0x00000000
 Txn Stamp: 388/0, Reference Count: 0.
 Txn Flags: Spooled
 Txn Commit Time: (1185848783) 2007/07/31 04:26:23
 Txn Size in Queue: 185
 First Buf's (0x123e89f8) Queue Flags: Spooled, Resident
 First Buf's Buffer Flags: Control
 NeedAck: Waiting for Acks from <[0009]>
 Traverse handle (0x11df9230) for thread CDRNsT2932 at txn (0x12482018) End_of_Q,  Flags: None
 Traverse handle (0x11df9230) for thread CDRNsT2932 at txn (0x12482018) End_of_Q,  Flags: None
 Traverse handle (0x123bc018) for thread CDRACK_0 at Head_of_Q,  Flags: None
 Traverse handle (0x1245c018) for thread CDRACK_1 at Head_of_Q,  Flags: None
```

The sysmaster SQL interface provides the syscdrctrl_txn and the syscdrctrl_buf tables to query the existing control messages queued in the system.
Example 6-23 shows the output taken from the same server state where the **onstat -g rqm** command output has been taken. You can see several matches of the values.

Example 6-23 sysmaster and control queues from the RQM

```
select * from syscdrctrl_buf


cbflags        2
cbsize         185
cbkeyserverid  2930
cbkeyid        0
cbkeypos       411
cbkeysequence  0
cbgroupid      1
cbcommittime   1185848783

select * from syscdrctrl_txn

ctkeyserverid  2930
ctkeyid        0
ctkeypos       411
ctkeysequence  0
ctstamp1       388
ctstamp2       0
ctcommittime   1185848783
ctuserid       0
ctfromid       0
```

6.4.4 Network interface statistics

The **onstat -g nif** command prints statistics about the network interface. 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 determine whether or not a site is in a hung state if it is not sending or receiving bytes. The NIF component exists on both sides, and the statistical output generated for each by **onstat -g nif** is similar.

Example 6-24 shows a snapshot of the **onstat -g nif** command's output on the g_srv1 server.

Example 6-24 onstat -g nif

```
$onstat -g nif
IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:09:13 -- 71904 Kbytes
```

```
NIF anchor Block: 11ed6810
        nifGState              RUN
           RetryTimeout        300

CDR connections:
 Id    Name              State              Version       Sent   Received
---------------------------------------------------------------------------
2932 singapore          RUN                       9     290298      55832
2934 frankfurt          RUN                       9     276226      84343
```

This command lists servers that are directly connected to the current server. The most important information in the **onstat -g nif** command's output is the server state. The important information in the State field is as follows:

- In the NIF statistics output, valid values for State are INIT, INTR, ABORT, SHUT, SLEEP, RUN, STOP, TIMEOUT, BLOCK, and SUSPEND.
- If the site is operational and active, then State will show a RUN status. If you see a BLOCK state, then there is a possibility of increased replication latency. What this means is that the target server is not able to keep up with the activity at the replication source server. In this case, it is time to reconfigure a larger server to improve performance.
- SUSPEND is shown whenever Enterprise Replication has been suspended by using the **cdr suspend server** command.

When CDR is stopped on the server using **cdr stop**, the State temporarily shows INTR, ABORT.

To get more specific information about the communication between the current server and another server in Enterprise Replication, use the **nif** command modifier server_id. Example 6-25 shows sample output from the command **onstat -g nif 2932**.

Example 6-25   onstat -g nif 2932

```
$onstat -g nif 2932

IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:09:34 -- 71904 Kbytes

NIF anchor Block: 11ed6810
        nifGState              RUN
           RetryTimeout        300

Detailed Site Instance Block: 11ebe6e0
        siteId                 2932
        siteState              256 <RUN>
        siteVersion            9
        siteCompress           0
        siteNote               0
        Send Thread            52 <CDRNsT2932>
        Recv Thread            87 <CDRNr2932>
```

```
Connection Start        (1185856991) 2007/07/31 06:43:11
Last Send               (1185857538) 2007/07/31 06:52:18
Idle Timeout            <Forever>
Flowblock               Sent 0 Receive 0
NifInfo: 11ceb2c0
        Last Updated    (1185856991) 2007/07/31 06:43:11
        State           Connected
        Total sent      303177 (28258042 bytes)
        Total recv'd    58393 (2567124 bytes)
        Retry           0 (0 attempts)
        Connected       1
Protocol                asf
Proto block             11612ab8
    assoc               10d69048
    state               0
    signal              0
Recv Buf                0
Recv Data               0
Send Count              0
Send Avail              0
```

Example 6-25 on page 286 shows detailed NIF statistics for server group ID 2932. If there is a non-zero value for Flowblock, there is a possibility of increased replication latency. This could be a potential problem, or the system may already be showing some symptoms. The reason for this is that the replication target server is not able to keep up with the transaction activity at the source server. If the replication target server receives more data than it can handle, it will signal the source server to stop sending data until further notice.

6.5 Monitoring the target replication server

After pushing the transactions out to the network, the transaction has left the scope of the source server. So now we look at the target side, with a focus on the monitoring possibilities of the receive queue. However, because the first component on the target is the NIF, it is not really different from the source side.

6.5.1 Reliable Queue Manager (RQM) statistics on the target

After the transaction package arrives on the target, it has to be maintained. Similar to the send queue on the source side, there is a receive queue on the target. This queue maintains all incoming transactions until the data sync component has applied the data to the table on the target replication server. There are requirements for the receive queue that are similar to the ones for the send queue. Reaching certain memory limits on the receive side causes spooling. This means additional utilization of disk space, which should not be exhausted.

Receive Queue statistics

The Receive Queue option of **`onstat`** reports the statistics related to the receive queue. Example 6-26 shows a sample output from the **`onstat -g rqm recvq`** command.

Except for a few differences, the receive queue statistics are similar to the send queue statistics. One difference between sendq and recvq is that Sent is the last transaction received from the source server for the given replicate ID.

Except in rare instances, such as replicating large transactions, spooling activity in the receive queue should never be seen. If it is seen, look for the root cause of the problem. Otherwise, an increase in replication latency may be the result.

Example 6-26 onstat -g rqm recvq

```
IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 03:05:02 -- 51904 Kbytes

CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0x11e60018) trg_receive
 Transaction Spool Name: trg_receive_stxn
 Insert Stamp: 83907/0
 Communal Stamp: 83907/0
 Flags: RECV_Q, SPOOLED, PROGRESS_TABLE
 Txns in queue:            2
 Txns in memory:           2
 Txns in spool only:       0
 Txns spooled:             0
 Unspooled bytes:          0
 Size of Data in queue:    2014106 Bytes
 Real memory in use:       2014106 Bytes
 Pending Txn Buffers:      0
 Pending Txn Data:         0 Bytes
 Max Real memory data used: 3370058 (4194304) Bytes
 Max Real memory hdrs used  2037716 (4194304) Bytes
 Total data queued:        78290540 Bytes
 Total Txns queued:        83853
 Total Txns spooled:       0
 Total Txns restored:      0
 Total Txns recovered:     0
 Spool Rows read:          0
 Total Txns deleted:       83851
 Total Txns duplicated:    0
 Total Txn Lookups:        736447

 Progress Table:
        Progress Table is Stable
                On-disk table name............:         spttrg_receive
                Not keeping dirty list.

Server    Group Bytes Queued      Acked                   Sent
------------------------------------------------------------------------------
  2930 0xb72004e          1406074 b72/1c3/333388/0        -       b72/1c5/27d0a8/0
  2930 0xb72004e                0 b78/7c/2da3d0/0         -        b78/7c/2da3d0/2
  2930 0xb720049                 0 b72/66/1731fc/0
-efffffff/efffffff/efffffff/efffffff

 First Txn (0x11e5fee0) Key:  2930/453/0x0027c6b8/0x00000000
 Txn Stamp: 83906/0, Reference Count: 1.
 Txn Flags: None
 Txn Commit Time: (1185857676) 2007/07/31 06:54:36
 Txn Size in Queue: 2014000
 First Buf's (0x1257df78) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 Traverse handle (0x1224b0d0) for thread CDRD_15 at txn (0x11e5fee0):
2930/453/0x0027c6b8/0x00000000
 Flags: In_Transaction

 Last Txn (0x12df0af0) Key:  2930/453/0x0027d0a8/0x00000000
 Txn Stamp: 83907/0, Reference Count: 0.
 Txn Flags: None
 Txn Commit Time: (1185857676) 2007/07/31 06:54:36
 Txn Size in Queue: 106
 First Buf's (0x12df0c30) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 Traverse handle (0x123f62b8) for thread CDRD_5 at txn (0x12df0af0) End_of_Q,  Flags: Communal

 Traverse handle (0x123f62b8) for thread CDRD_5 at txn (0x12df0af0) End_of_Q,  Flags: Communal
 Traverse handle (0x123fd018) for thread CDRD_5 at Head_of_Q,  Flags: None
 Traverse handle (0x12269018) for thread CDRD_7 at Head_of_Q,  Flags: Communal
 Traverse handle (0x1226b018) for thread CDRD_7 at Head_of_Q,  Flags: None
 Traverse handle (0x122f8070) for thread CDRD_8 at Head_of_Q,  Flags: Communal
 Traverse handle (0x1230d018) for thread CDRD_8 at Head_of_Q,  Flags: None
 Traverse handle (0x11e4d230) for thread CDRNrA2930 at Head_of_Q,  Flags: None
 Traverse handle (0x11e68330) for thread CDRD_9 at Head_of_Q,  Flags: Communal
 Traverse handle (0x122de118) for thread CDRD_9 at Head_of_Q,  Flags: None
 Traverse handle (0x1257d4e0) for thread CDRD_10 at Head_of_Q,  Flags: Communal
 Traverse handle (0x125764e0) for thread CDRD_10 at Head_of_Q,  Flags: None
 Traverse handle (0x12339118) for thread CDRD_12 at Head_of_Q,  Flags: Communal
 Traverse handle (0x11b7e0d0) for thread CDRD_12 at Head_of_Q,  Flags: None
 Traverse handle (0x11808610) for thread CDRD_13 at Head_of_Q,  Flags: Communal
 Traverse handle (0x12050018) for thread CDRD_13 at Head_of_Q,  Flags: None
 Traverse handle (0x122bf018) for thread CDRD_14 at Head_of_Q,  Flags: Communal
 Traverse handle (0x12579018) for thread CDRD_14 at Head_of_Q,  Flags: None
 Traverse handle (0x1224b0d0) for thread CDRD_15 at txn (0x11e5fee0):
2930/453/0x0027c6b8/0x00000000
 Flags: In_Transaction
 Traverse handle (0x1219e118) for thread CDRD_15 at Head_of_Q,  Flags: None
 Traverse handle (0x1233d018) for thread CDRD_17 at Head_of_Q,  Flags: Communal
 Traverse handle (0x1219f410) for thread CDRD_17 at Head_of_Q,  Flags: None
```

Monitoring the receive queue with the sysmaster database interface in a balanced workload should commonly show only a few entries. Similar to all other RQM maintained queues, the sysmaster provides two tables for selecting status information for the receive queue: the syscdrrecv_txn and syscdrrecv_buf tables. When transactions arrive on the target side and are not already processed by the data sync component, the output looks similar to the output in Example 6-27.

Example 6-27 sysmaster interface output for the receive queue

```
$echo "select * from syscdrrecv_txn" | dbaccess -e  sysmaster

ctkeyserverid  2930
ctkeyid        351
ctkeypos       365572
ctkeysequence  0
ctstamp1       10711
ctstamp2       0
ctcommittime   1185857050
ctuserid       0
ctfromid       2930

ctkeyserverid  2930
ctkeyid        351
ctkeypos       369140
ctkeysequence  0
ctstamp1       10712
ctstamp2       0
ctcommittime   1185857050
ctuserid       0
ctfromid       2930
```

The syscdr database stores spooled transaction information for the receive queue rows in the syscdr:trg_recv_stxn table. As previously pointed out, a spool on the receive side is seldom seen. During the testing for this book, there have never been entries in this table.

6.5.2 Receive Manager statistics

The Receive Manager is the module that operates between the receive queue and the data sync on the target system.

Sample output from Receive Manager statistics is displayed in two parts, as shown in Example 6-28 and Example 6-29 on page 292.

Example 6-28 onstat -g rcv full (part 1)

```
$onstat -g rcv full
IBM Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 00:09:26 -- 111904 Kbytes
ServerId: 2930
Flags 0x4
ServerId: 2936
Flags 0x4

Threads:
         Id Name         State  Handle
         57 CDRACK_1       Idle 11bc1ef8
         56 CDRACK_0       Idle 11baca28

 Receive Manager global  block 117f6018
         cdrRM_inst_ct:                          2
         cdrRM_State:                   00000000
         cdrRM_numSleepers:             3
         cdrRM_DsCreated:               11
         cdrRM_MinDSThreads:            3
         cdrRM_MaxDSThreads:            12
         cdrRM_DSBlock                  0
         cdrRM_DSParallelPL             1
         cdrRM_DSFailRate               0.047395
         cdrRM_DSNumRun:                43232
         cdrRM_DSNumLockTimeout         1
         cdrRM_DSNumLockRB              193
         cdrRM_DSNumDeadLocks           1751
         cdrRM_DSNumPCommits            0
         cdrRM_ACKwaiting               17
         cdrRM_totSleep:                292
         cdrRM_Sleeptime:               814
         cdrRM_Workload:                4
         cdrRM_optscale:                4
         cdrRM_MinFloatThreads:         2
         cdrRM_MaxFloatThreads:         7
         cdrRM_AckThreadCount:          2
         cdrRM_AckWaiters:              2
         cdrRM_AckCreateStamp:          Tue Jul 31 06:39:42 2007
         cdrRM_DSCreateStamp:           Tue Jul 31 06:45:54 2007
         cdrRM_acksInList:              0
         cdrRM_BlobErrorBufs:           0
```

Some of the terms from the **onstat -g rcv full** command output in Example 6-28 on page 291 are as follows:

- cdrRM_DSFailRate: If the fail rate value is non-zero, then the apply component performance might suffer. This can be due to lock errors while applying data. Non-zero does not mean that replicated transactions were aborted. The problem might get corrected internally and the transaction might have been re-applied.
- cdrRM_DSNumLockTimeout: Number of lock timeouts.
- cdrRM_DSNumDeadLocks: Number of deadlock errors.
- cdrRM_DSParallelPL: If non-zero, then data sync might be serializing the apply process.

Part 2 of the **onstat -g rcv full** command output shows parallelism statistics and the source server statistics, as shown in Example 6-29.

Example 6-29 onstat -g rcv full (part 2)

```
Receive Parallelism Statistics
Server Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
  2930  42937         0      0   4496      4  391.15    1.24   78.21
  2936    295        93      1    232      2  100.18    1.84   32.67

Tot Pending:93    Tot Active:1  Avg Pending:388.54  Avg Active:1.24
Commit Rate:78.74

Time Spent In RM Parallel Pipeline Levels
Lev. TimeInSec  Pcnt.
   0       145  27.10%
   1       230  42.99%
   2       160  29.91%

Statistics by Source

Server 2930
Repl        Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
192020558   42937  268204  269206       0 2007/07/31 06:48:52 2007/07/31 06:48:52

Server 2936
Repl        Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
192020558     294     444   10576       0 2007/07/31 06:48:52 2007/07/31 06:48:50
```

The server data displayed in Example 6-29 are:

- Receive Manager parallelism statistics:
  - Server: Source server ID.
  - Tot.Txn: Total transactions received from this source server.
  - AvgAct: Average number of transactions applied in parallel from this source server.

  - CommitRt: Number of transactions applied per second from this source server.
  - Avg Active: Average number of transactions that data sync is applying in parallel.
  - Commit Rate: Number of transactions applied per second.
- Statistics by Source

  Here you get to see per source server and per replicate statistics:

  - Repl: Replicate ID. Get the replicate name and table name from the output of the **onstat -g cat repls** command.
  - Txn: Total number of transactions applied.
  - Ins: Number of inserts received from this source server.
  - Del: Number of deletes received from this source server.
  - Upd: Number of updates received from this source server.
  - Last Target Apply: Last transaction apply time.
  - Last Source Commit: Last transaction commit time at the source server.
  - Replication Latency: Last Target Apply minus the Last Source Commit.

6.5.3 Data Sync statistics

Data Sync is the subsystem in Enterprise Replication where the transactions are finally applied to the target table. There are multiple threads in the system, which can process incoming transactions in parallel. But each thread applies one transaction at a time. Statistics about this module can be seen by running the **onstat -g dss** command; a sample output is shown in Example 6-30.

Example 6-30 onstat -g dss

```
Informix Dynamic Server Version 11.50.UC1     -- On-Line -- Up 02:55:59 -- 51904 Kbytes
DS thread statistic

cmtTime      Tx       Tx        Tx      Last Tx
Name         < local  Committed Aborted Processed  Processed Time
------------ -------  --------- ------- ---------  -----------------
CDRD_5             0      4560       0      4560  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9
CDRD_6             0      4451       0      4451  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9
CDRD_7             0      4435       0      4435  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9
CDRD_8             0      3957       0      3957  (1185857139) 06:45:39
        Tables (0.0%): informix:test@stores9 informix:cdr_deltab_000043@stores9
        Databases: stores9
CDRD_9             0      1676       0      1676  (1185857139) 06:45:39
        Tables (0.0%): informix:test@stores9 informix:cdr_deltab_000043@stores9
        Databases: stores9
CDRD_10            0       691       0       691  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9
CDRD_11            0       616       0       616  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9
CDRD_12            0       653       0       653  (1185857139) 06:45:39
        Tables (0.0%):
        Databases: stores9

CDR_DSLOCKWAIT = 5
CDR_DSCLOSEINTERVAL = 60
```

The output in Example 6-30 on page 293 shows that there are eight Data Sync threads running in parallel. Transactions handled by each thread are listed in the tabular output.

In this output, it is important to make sure that transaction processing is evenly distributed across all Data Sync threads. If you see most of the transaction processing is done by one Data Sync thread, then for some reason data is getting applied serially, which can increase replication latency. Look at the **`onstat -g rcv full`** command output for further help in diagnosing the problem.

In addition, the CDR_DSLOCKWAIT should match with the CDR_DSLOCKWAIT config parameter value in the `onconfig` file.

6.6 Automated monitoring

OAT makes extensive use of the sysmaster database to provide information. If you are developing a system that requires automating monitoring, such as an embedded application, you probably need to make use of the sysmaster database as well.

Note: It is not a good idea to query the syscdr database directly because that can cause locking conflicts with Enterprise Replication itself, which in turn can actually cause problems.

The schema for sysmaster is located in `$INFORMIXDIR/etc/sysmaster.sql`. Look at that file for the column descriptions. In this section, we point out which sysmaster table can be used to monitor which component of ER. For the table and column descriptions, see the `sysmaster.sql` file.

Note: The sysmaster database contains pseudo tables that map to memory structures. That means that the contents of a sysmaster table may change each time that it is selected.

6.6.1 Key tables to monitor in sysmaster

There are many tables in sysmaster related to ER. In this section, we discuss the most useful ones. Figure 6-17 illustrates the relationship of these key tables to ER.

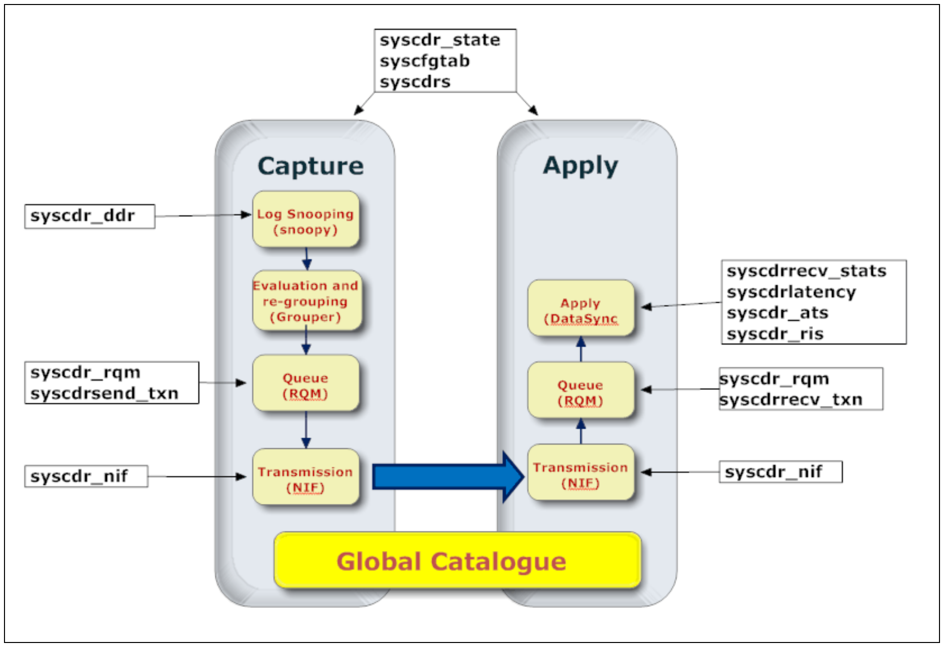


Figure 6-17 Relationship with key tables concerning ER in sysmaster

- syscdr_state

  This table can be used to monitor the ER state in general. It will display if the system is active or not.

- syscfgtab

  This table is not specifically an ER table in sysmaster, but is highly useful for ER monitoring. It contains the configuration items that are active for the Informix server.

- syscdr_ddr

  This table contains snoopy information that can be useful in determining the risk of a log wrap.
- syscdrs

  This table displays information about each of the cdr servers that exist in the domain.

  Note: Leaf servers only know about themselves and their parent.
- syscdr_rqm

  This table displays information about the queue. There is one row for each of the queues (that is, send, receive, control, ack, and so on.)
- syscdrsend_txn, syscdrack_txn, and syscdrctrl_txn

  These tables display information about the transaction headers that are in the rqm queue and currently in memory.
- syscdrrecv_stats

  This table displays statistics of transactions that have been applied from each source system.
- syscdrlatency

  This table displays detailed information about the data being received from each node.
- syscdr_nif

  This table displays network statistics from each of the servers to which we are connected.
- syscdr_ats/syscdr_ris

  This table displays a list of files that are contained in the ATS or RIS spooling directories.

6.7 Troubleshooting a running ER environment

In this section, we provide some suggestions and help for troubleshooting a running ER environment.

6.7.1 The ATS and RIS files and data inconsistency

Running a newly defined replicate in an ER environment successfully can only happen if the data sets are consistent, which means on all involved database servers that the data sets are either the same or the local data is not touched, and so changes on them are not distributed to other servers. Another influential factor for ensuring a consistent data set across the replication servers is, in addition to the choice of the right data flow definition, choosing the right conflict resolution strategy.

Choosing the wrong conflict resolution and causing inconsistencies in the databases can have a serious impact on the schedule of the DBA's daily work. Having to manually repair these databases can take significant time. It involves the investigation of which rows are affected, and communication with the application development for the decision on which is the final data and what should change with the existing data.

An indication of data inconsistency

Determining data inconsistency requires frequent monitoring of the `online.log` file or the directory where with the replication server definition of the RIS/ATS is saved. The `online.log` file reports the inconsistency on the target in a similar fashion, as shown in Example 6-31.

Example 6-31 ATS file warning written by the data sync thread on the target

```
04:07:31  CDR CDRD_15: transaction aborted (One or more rows in a transaction
defined with tx scope were rejected) with sql error 0 isam error 0.
04:07:31  CDR CDRD_15: failed transaction spooled to file
/files/ATS/ats.singapore.newyork.D_15.070726_04:07:31.2
```

Depending on the conflict resolution

The basis for our discussion here is the different data sets across the replication server. ER defines several conflict resolution strategies. Some of the strategies are better candidates for generating the RIS and ATS files than others. The ignore strategy is most likely the best candidate to generate the error files. With this strategy, the server will not try to do any kind of resolution. If you look at the compatibility matrix in the ignore strategy, the server never modifies the requested operation from one to another. Assume a situation where the source server has sent an insert request on a particular table for a certain primary key. Also assume the key already exists on the target. An RIS file will be written and the change is discarded for the target.

On the other hand, choosing Always-Apply does not mean that there will never be inconsistencies. Assume the DBA has defined a replicate with a frequency of every 10 minutes for distributing the data. On one server a delete occurs, on another server, a short time after the delete, there is an update on the same primary key. What is the expected result after the distribution of the changes? All targets have seen the delete, but one (the server that has initiated the delete itself) will finally have modified the update into a insert.

ATS and RIS files

Independent of why the ATS and RIS files are written, the general structure of the files is the same. There are the following components in an ATS file:

- If the RIS spooling is enabled, the ATS file will refer first to the RIS file.
- Additional information about target and source and the commit time of the transactions follow.
- Finally, detailed information about the row with the column values is written, depending on whether the query is dynamically or statically declared, as shown in Example 6-32.

Example 6-32 Sample ATS files

```
#aborted Insert
TXH RIS file:/tmp/ris.newyork.singapore.D_3.070725_07:13:13.5 has also been created for this
transaction
==========
TXH Source ID:2932 / Name:singapore / CommitTime:07-07-25 07:13:13
TXH Target ID:2930 / Name:newyork / ReceiveTime:07-07-25 07:13:13
TXH Number of rows processed when transaction was aborted:28
TXH All rows in a transaction defined with row scope were rejected
----------
RRH Row:1 / Replicate Id: 192020483 / Table: stores9@informix.customer / DbOp:Insert
RRD 1905|Ludwig|Pauli|All Sports Supplies|213 Erstwild Court||Sunnyvale|CA|94086|408-789-8075

#aborted Delete
TXH RIS file:/tmp/ris.newyork.frankfurt.D_3.070725_06:46:48.1 has also been created for this
transaction
```

```
==========
TXH Source ID:2934 / Name:frankfurt / CommitTime:07-07-25 06:46:47
TXH Target ID:2930 / Name:newyork / ReceiveTime:07-07-25 06:46:48
TXH Number of rows processed when transaction was aborted:1
TXH All rows in a transaction defined with row scope were rejected
----------
RRH Row:1 / Replicate Id: 192020486 / Table: stores9@informix.customer / DbOp:Delete
RRD 110|?|?|?|?|?|?|?|?|?
```

The RIS files are, most of the time, more detailed about the SQL errors that happen on a discarded operation. This situation is especially true for conflict resolution based on your own stored procedure. The return code of the SP for the discard is specified in the RIS. See Example 6-33 for a collection of sample RIS files with different reasons for why they have been written. Depending on the detailed row information, a DBA should be able to manually compare the rows on both sides and decide what has to be done to solve the inconsistency. To automate this activity, there is an option in the **`cdr`** utility to force synchronization between the servers.

Example 6-33 RIS files and their content

```
#Attempt to delete a not existing row
TXH Source ID:2934 / Name:frankfurt / CommitTime:07-07-25 06:49:49
TXH Target ID:2930 / Name:newyork / ReceiveTime:07-07-25 06:49:49
----------
RRH Row:1 / Replicate Id: 192020486 / Table: stores9@informix.customer / DbOp:Delete
RRH CDR:7 (Error: Delete aborted, row does not exist in target table) / SQL:0 / ISAM:0
RRD 101|?|?|?|?|?|?|?|?|?

#Warning row did exists, moved the insert into an update
TXH Source ID:2932 / Name:singapore / CommitTime:07-07-25 07:21:04
TXH Target ID:2930 / Name:newyork / ReceiveTime:07-07-25 07:21:04
----------
RRH Row:1 / Replicate Id: 192020487 / Table: stores9@informix.customer / DbOp:Insert
RRH CDR:2 (Warning: Insert exception, row already exists in target table, converted to update)
/ SQL:0 / ISAM:0
LRS 2934 (frankfurt)|1185340864 (07/07/25 07:21:04)
LRD 757|Ludwig|Pauli|All Sports Supplies|213 Erstwild Court||Sunnyvale|CA|94086|408-789-8075
RRS 2932 (singapore)|1185340864 (07/07/25 07:21:04)
RRD 757|Ludwig|Pauli|All Sports Supplies|213 Erstwild Court||Sunnyvale|CA|94086|408-789-8075

#Insert failed -- row already there
TXH Source ID:2932 / Name:singapore / CommitTime:07-07-25 07:13:15
TXH Target ID:2930 / Name:newyork / ReceiveTime:07-07-25 07:13:15
----------
RRH Row:1 / Replicate Id: 192020483 / Table: stores9@informix.customer / DbOp:Insert
RRH CDR:5 (Error: Insert aborted, row already exists in target table) / SQL:0 / ISAM:0
LRD 2829|Ludwig|Pauli|All Sports Supplies|213 Erstwild Court||Sunnyvale|CA|94086|408-789-8075
RRD 2829|Ludwig|Pauli|All Sports Supplies|213 Erstwild Court||Sunnyvale|CA|94086|408-789-8075
```

The reasons for the existence of the RIS files described above are quite obvious; they are created because of inconsistencies in the data. But there are also other reasons for the occurrence of the RIS files. Assume that a non-mastered replicate was defined and enabled across the replication servers with quite different schema definitions for the tables. Applying changes on the target, which come from a source with a completely different layout, could potentially cause corruption or some RIS files might appear immediately after the first changes on the data are applied, as shown in Example 6-34.

Example 6-34 RIS file with tables with different schema definitions

```
TXH Source ID:2934 / Name:frankfurt / CommitTime:07-07-25 19:37:18
TXH Target ID:2936 / Name:london / ReceiveTime:07-07-25 19:37:35
----------
RRH Row:1 / Replicate Id: 192020490 / Table: stores9@informix.customer / DbOp:Update
RRH CDR:45 (Error: Could not read from received stream.) / SQL:0 / ISAM:0
RRD ????Unable to format row data
RRS C001 00000b72
```

6.7.2 Monitoring the RQM sendq queue

Periodically monitoring the send and the receive queues in an ER environment is one the few required DBA activities. There are reasons for this requirement. For example, spooling of the send transactions has a negative impact of the I/O throughput and could be an indication of a problem on one of the target replication servers.

In a balanced system, where all servers are available, there are no scheduled replication frequencies, and there is reasonable network performance, the ER replication servers are able to provide fast throughput of the distributed data. But there are many parameters in the database server, in the network, and in the operating system, that can hamper throughput. Looking at the database server, there are a number of influential parameters that can cause a transaction spool. For example:

- The target replication server is not available.
- Replicates are defined in ER with specific distribution times.
- Inconsistencies on the target can produce ATS/RIS files, which can slow down the apply of the rows.
- Parallel peak situations on the target.

In a production environment, there should be sufficient space configured for spooling the transaction headers in a normal dbspace and the attached sblobs in an sbspace. But in case the sizing was based on a different workload and the dbspaces start filling up, it should not get to the point where the sbspace is completely filled. In that situation, ER would block the server, the connected user sessions will not be able to write to the server, and the system would hang. Adding chunks to the sblob space that runs out of disk space will also not help, and a reboot of the database server is required.

To avoid this situation, make frequent checks of the **`onstat -g rqm sendq`** outputs, as shown in Example 6-35. You can see the spool by looking at the values for `Txns in spool only`. In the test case scenario, which is monitored here, the number of spooled transactions increases slightly with each **`onstat`** call.

Example 6-35 onstat -g rqm sendq

```
#run frequently onstat -g rqm sendq , look at the Txns spooled line
#first run
RQM Statistics for Queue (0x11dbd018) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 32637/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:             13448
 Log Events in queue:       1
 Txns in memory:            387
 Txns in spool only:        13061
 Txns spooled:              13061
 Unspooled bytes:           204652
 Size of Data in queue:     7127512 Bytes

#second run
RQM Statistics for Queue (0x11dbd018) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 38786/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:             19573
 Log Events in queue:       0
 Txns in memory:            386
 Txns in spool only:        19187
 Txns spooled:              19214
 Unspooled bytes:           190270
 Size of Data in queue:     10374220 Bytes

#third run
RQM Statistics for Queue (0x11dbd018) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 58348/0
```

```
Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
Txns in queue:              39063
Log Events in queue:        2
Txns in memory:             387
Txns in spool only:         38676
Txns spooled:               38676
Unspooled bytes:            201014
Size of Data in queue:      20704064 Bytes
Real memory in use:         204194 Bytes
Pending Txn Buffers:        20
Pending Txn Data:           2120 Bytes
```

In addition to the **onstat -d** command, the **onstat -g rqm** command provides some basic output options for monitoring the use of the sbspaces used by the queuer, as shown in the output of Example 6-36.

Example 6-36 onstat -g rqm sbspaces

```
#first run
RQM Space Statistics for CDR_QDATA_SBSPACE:
-------------------------------------------
name/addr      number    used       free       total      %full   pathname
0x10d98a40     2         16153      27908      44061      0
/chunks/IDS11/chunk_prim_sblob
cdr_sbspace    2         16153      27908      44061      0

#second run
RQM Space Statistics for CDR_QDATA_SBSPACE:
-------------------------------------------
name/addr      number    used       free       total      %full   pathname
0x10d98a40     2         18142      25919      44061      0
/chunks/IDS11/chunk_prim_sblob
cdr_sbspace    2         18142      25919      44061      0

#third run
RQM Space Statistics for CDR_QDATA_SBSPACE:
-------------------------------------------
name/addr      number    used       free       total      %full   pathname
0x10d98a40     2         38824      243        39067      15900
/chunks/IDS11/chunk_prim_sblob
0x12877e38     3         275        46330      46605      0       chunk_prim_sblob_add
cdr_sbspace    2         39099      46573      85672      0
```

But investigation of the spool activity is not sufficient for problem analysis. Assume that there is an update-anywhere scenario where a certain number of servers are defined on the target side. You have to manually look at each one to determine which one could cause the spool.

Look at the output shown in Example 6-37. Here you can see that there is only one target server causing the spool. In this particular test case, it is quite obvious that the server with the groupid 2932 (singapore) causes the queuing.

Example 6-37 Investigate the target causing the spool

```
$echo "select * from syscdrq" | dbaccess -e sysmaster

srvid      2932
repid      192020559
srcid      2930
srvname    singapore
replname   raadr
srcname    newyork
bytesqued  1049690

srvid      2934
repid      192020559
srcid      2930
srvname    frankfurt
replname   raadr
srcname    newyork
bytesqued  370
```

If the spooling cannot be stopped, the `online.log` file reports the messages shown in Example 6-38. The server issues a DDR block and a reboot is required.

Example 6-38 Server runs out of sbspace for the spool

```
06:44:06  Logical Log 605 Complete, timestamp: 0xca4898.
06:44:12  Logical Log 606 Complete, timestamp: 0xca99fe.
06:44:12  CDR WARNING:RQM sbspace cdr_sbspace is full, Please add the space.
error number = 131
06:44:12  CDR QUEUER: Send Queue space is FULL - waiting for space in
CDR_QDATA_SBSPACE
06:44:16  Logical Log 607 Complete, timestamp: 0xcae171.
06:44:19  Logical Log 608 Complete, timestamp: 0xcb2bcf.
06:44:21  Logical Log 609 Complete, timestamp: 0xcb5f0d.
06:44:24  Logical Log 610 Complete, timestamp: 0xcb9e57.
06:44:27  Logical Log 611 Complete, timestamp: 0xcbec7e.
06:44:29  Logical Log 612 Complete, timestamp: 0xcc1a74.
```

6.8 Using cdr view

One of the problems with **onstat** is that it displays information from only one server at a time. For ER, this is not good enough. Because there are multiple nodes within the ER domain, we need to have a way to view information about the entire domain with a single command.

The **cdr view** command provides this functionality. It reads the syscdr metadata, attaches to each of the servers within the ER domain, and then displays some key information about each of the nodes within the domain.

6.8.1 Viewing the status of the servers

The **cdr view state** command displays a one line state of each of the nodes and displays the status of each of the components of ER on each of those nodes.

Example 6-39 cdr view stat

```
$cdr view state
STATE
Source  ER               Capture          Network          Apply
        State            State            State            State
---------------------------------------------------------------------------
g_serv1 Active           Running          Running          Running
g_serv2 Shut Down        Uninitialized    Down             Uninitialized
g_serv3 Active           Running          Running          Running
```

6.8.2 Viewing network traffic

You can view the network statistics by running **cdr view nif**, as shown in Example 6-40.

Example 6-40 View network traffic

```
$cdr view nif
NIF
Source  Peer    State         Messages  Messages Messages  Transmit
                                  Sent  Received  Pending      Rate
---------------------------------------------------------------------------
g_serv1 g_serv2 Connected      8863371   2957198        0 11507895.000
        g_serv3 Connected      8861272   2957094        0 28620.289
g_serv2 g_serv1 Connected        22111     67257        0 87398.984
        g_serv3 Connected           64        65        0    59.595
g_serv3 g_serv1 Connected      2957198   8861168        0 28618.486
        g_serv2 Connected         8262      8244        0  8307.312
```

6.8.3 Viewing the node profiles

The **`cdr view profile`** command provides similar information as the OAT node detail window that is shown in Figure 6-6 on page 256. This is often an excellent starting point for isolating a problem and should probably be included with any case that is opened with tech support. It provides a summary of each of the nodes in the entire ER domain.

Example 6-41 cdr view profile

```
$cdr view profile
ER PROFILE for Node g_serv1          ER State Active

DDR - Running                        SPOOL DISK USAGE
  Current       711:8146944            Total                256000
  Snoopy        711:8144768            Metadata Free         12838
  Replay        711:7352344            Userdata Free        238693
  Pages from Log Lag State    86011
                                     RECVQ
SENDQ                                  Txn In Queue              0
  Txn In Queue             41          Txn In Pending List       0
  Txn Spooled               0
  Acks Pending              0        APPLY - Running
                                       Txn Processed            70
NETWORK - Running                      Commit Rate            0.00
  Currently connected to 2 out of 2    Avg. Active Apply      2.14
  Msg Sent            17959286         Fail Rate              0.00
  Msg Received         5992608         Total Failures            0
  Throughput          28842.41         Avg Latency            0.00
  Pending Messages          0          Max Latency               0
                                       ATS File Count            0
                                       RIS File Count            0
---------------------------------------------------------------------------
ER PROFILE for Node g_serv2          ER State Active

DDR - Running                        SPOOL DISK USAGE
  Current       257:1122304            Total                256000
  Snoopy        257:1120148            Metadata Free         12838
  Replay        257:24                 Userdata Free        238693
  Pages from Log Lag State    87726
                                     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         61061
NETWORK - Running                      Commit Rate          240.40
  Currently connected to 2 out of 2    Avg. Active Apply      1.00
  Msg Sent               61450         Fail Rate              0.00
  Msg Received          184741         Total Failures            0
  Throughput          37150.13         Avg Latency            1.00
  Pending Messages          0          Max Latency               1
                                       ATS File Count            0
                                       RIS File Count            0
---------------------------------------------------------------------------
```

```
ER PROFILE for Node g_serv3          ER State Active

DDR - Running                        SPOOL DISK USAGE
  Current       256:13963264           Total                256000
  Snoopy        256:13961140           Metadata Free         12838
  Replay        256:24                 Userdata Free        238693
  Pages from Log Lag State    84591
                                     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       2987076
NETWORK - Running                      Commit Rate           93.55
  Currently connected to 2 out of 2    Avg. Active Apply      2.44
  Msg Sent            3004725          Fail Rate              0.00
  Msg Received        8986860          Total Failures            0
  Throughput         14429.27          Avg Latency            0.00
  Pending Messages          0          Max Latency               0
                                       ATS File Count            0
                                       RIS File Count            0
---------------------------------------------------------------------------
```

6.9 Miscellaneous issues

We have looked at the normal information that we might want to monitor, and we have also explained some of the things that can be one if these issues occur. However, there are still some things that have to be understood that just do not fit into the normal classifications. This section deals with those issues.

6.9.1 Handling asynchronous communication in ER

All operations in ER are asynchronous in nature, which includes the control messages and data replication. Control messages are messages sent across servers that modify such elements as systems information and replicate definitions. These messages are primarily generated by the operations performed by **cdr** commands, and those operations that modify the syscdr database.

Asynchronous control messages can cause some confusion. When an administrative operation is performed using Enterprise Replication, the status that returns from those operations is indicative only of the success or failure of the operation at the database server to which commands have been directed. However, the operation might still be propagating through the other ER database servers in the network at that time.

Due to this asynchronous propagation, it may seem that the operation was successful and that the next command, which may depend on the previous command, can be performed immediately. But this action could result in an error, which may be surprising. Consider the four commands shown in Example 6-42, which define four servers in a topology that is run from a script.

Example 6-42 Defining four nodes in ER

```
cdr define server -c srv1    -i 500 -I g_srv1
cdr define server -c srv4 -L -i 500 -S g_srv1 -I g_srv4
cdr define server -c srv2 -N -i 500 -S g_srv1 -I g_srv2
cdr define server -c srv3 -L -i 500 -S g_srv2 -I g_srv3
```

Depending on the connectivity and computing power of individual nodes, the sequence of commands in Example 6-42 may produce the output shown in Example 6-43 (when the last command is executed).

Example 6-43 ER command failed due to asynchronous messaging

```
$ cdr define server -c ws306s1 -i 500 -L -S g_srv2 -I g_srv3
command failed -- invalid server  (47)
```

Another example of an error is one that happens when an ER node is dropped and recreated. This is depicted in Example 6-44.

Example 6-44 Failure due to dropping and recreating a node immediately

```
$ cdr delete server -c srv1 g_srv1
$ cdr define server -c srv1 -i 500 -I g_srv1
command failed -- The syscdr database is missing!
 (115)
```

Due to this asynchronous propagation, you should avoid performing control operations in quick succession that might directly conflict with one another, without verifying that the first operation has successfully propagated through the entire enterprise network. Specifically, avoid deleting Enterprise Replication objects, such as replicates, replicate sets, and Enterprise Replication servers, and immediately re-creating those objects with the same name. Doing so can cause failures in the Enterprise Replication system at the time of the operation or perhaps even later.

These failures might manifest themselves in ways that do not directly indicate the source of the problem. If you must delete and re-create a definition, use a different name for the new object. For example, delete replicate a.001 and re-create it as a.002, or wait until the delete action has successfully propagated through the entire Enterprise Replication system before you re-create the object.

The former strategy is especially appropriate if you have database servers that are not connected to the Enterprise Replication network at all times. It could take a significant amount of time before the operation is propagated to all those disconnected servers.

6.9.2 Increase in replication latency

There can also be an increase in latency due to machine clock synchronization issues. Always make sure that the machine clocks are in sync between replication servers. Otherwise, there is a possibility of increased replication latency when using a time stamp or stored procedure conflict resolution rule.

If clocks are out of sync, the server prints the following warning message in the server message log file:

```
Warning - The operating system time of day clock at the remote site differs
from the local site.
```

This difference can slow down the data-sync apply performance at the target server. But it happens only if time stamp or stored procedure conflict resolution rules are being used.

6.9.3 Data not being replicated

In this scenario, a replicate was defined and replication was started, but the data is not being replicated.

Make sure that the replicate is defined and active at all replicate participants. At each replicate participant, run the following commands and make sure that replicate status shows it as `active`:

- **`Cdr list repl`**: Shows the replicate status in the syscdr database.
- **`Onstat -g cat repls`**: Shows the replicate status in memory.

It is possible that the **`cdr define replicate`** and **`cdr start replicate`** commands succeed at the local server but fail at remote servers. These peer server failure messages can be retrieved using the **`cdr error`** command.

If the CDR control command fails at a remote server, in the local server message log file, you will see a message such as:

```
13:15:16  CDR GC peer processing failed: command: start repl, error 100, CDR
server 200.
```

7

MACH11 clusters

IBM Informix MACH11 describes a group of Informix database cluster solutions. In this configuration, a primary database server replicates all changes on the local maintained data set to a single or a number of standby database servers. The existing secondary servers can be updatable or read-only. Each solution targets a particular cluster data availability and client workload balancing goal. Because these solutions can be combined together, powerful database server environments can be created easily and are transparent in their usage.

High Availability Data Replication (HDR), Remote Secondary Standby (RSS), and Shared Disk Secondary (SDS) servers are fundamental components for building virtualized cloud, cluster, or network of cluster database server environments. In this chapter, we describe these functions, starting with a general discussion. We then discuss the following topics:

- A general discussion about database cluster environments, targets, and infrastructure layouts
- General functionality provided by the standby database server
- Failover management
- Cluster monitoring facilities provided by the informix database server

We provide details about the strength of each particular solution, their requirements for setup, and initialization in the chapters followed.

7.1 Introduction

Globalization is a term commonly used today. You can refer to globalization when you look at companies' business models: worldwide information exchange such as the internet, social networks, and global networks of countries exchanging natural resources, food, and goods. For all these global exchanges, a business needs a reflection of how all these data flow are maintained. The best solution for an infrastructure of database server resources supporting a globalized environment is a *cluster*. Database clusters combine resources independent of their physical location. The clusters can be grouped locally in one room, in one building, or can be grouped by servers in a country or continent. MACH11 database cluster solutions support data resilience with one to many standby copies in addition to resource management. MARC11 provides the flexibility in adjusting the deployed server resources to meet the current throughput requirements that are key to a successful cluster environment.

You can learn from the following sections about how to establish a powerful MACH11 cluster environment. The layout of your database environment should define a mapping that best matches your organization's needs. The environment contains all the secondary database server types that support your existing clients based on workload and functionality. You know how to add on demand resources to a cluster environment using a concrete understanding about which configuration adjustments need to be applied to the new resources. The definition of an appropriate failover strategy strengthens the continuous access to the data from outages.

7.1.1 Server types of a MACH11 cluster

MACH11 is a synonym for a group of specific standby database servers. These database servers are always attached to a primary database server. This primary server maintains the data set. Any changes on the data set are replicated to the attached standby servers by using log files. Access to the latest version of the data can be consistently provided by all the servers in a MACH11 environment. The standby servers can manage access requests of any types of similar client applications to a primary database server, depending on the configuration, with a few restrictions in view of functionality.

MACH11 database cluster environments can be built as the following Informix database server types:

- Remote Standby Secondary (RSS)
- Shared Disk Secondary (SDS)
- High Availability Data Replication (HDR)

In the following section, we discuss the MACH11 database cluster and give a short introduction of the available Informix secondary server types.

7.1.2 Shared Disk Secondaries: Building a database server cloud

The Informix Shared Disk Secondary (SDS) server has the configuration where one primary server owns the underlying storage layout. The storage layout is used by dbspaces and chunks administered by the database server. You can add and detach a number of secondary servers accessing the same storage in a shared all (except for temporary storage) approach. Because the standby server shares the dbspaces, there are no requirements in terms of time for cloning to add new resources. The SDS server continuously obtains the newest logical log information from the primary and is able to always provide the latest version of all your data. A simplified Informix SDS cluster environment is shown in Figure 7-1.

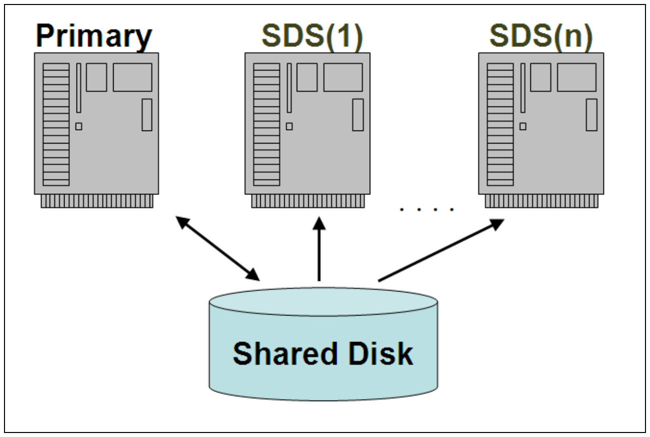


Figure 7-1 Simplified MACH11 SDS cluster topology

Choosing an SDS environment for a cluster enables you to adjust the available memory and processor resources while maintaining the same data to serve different workload requirements.

7.1.3 Remote Standby Secondary: Ensuring resilience

The RSS database cluster environment is the right solution if you need to focus on high availability for a master database server. You are allowed to set up a one-to-many replication between the primary and the secondary servers. All the secondary servers own their set of dbspaces as read only or updatable to ensure a complete fault tolerance against hardware and software outages.

RSS servers are a copy of the complete data set from the primary database server. Any changes applied to the primary are replicated by continuously shipping the logical logs to the associated secondary server.

Figure 7-2 shows a sample RSS topology.

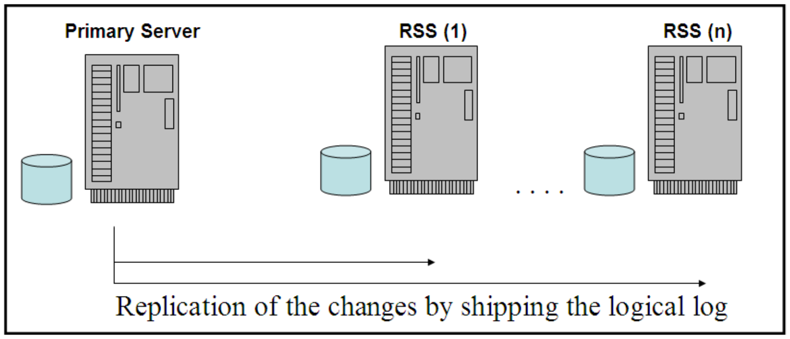


Figure 7-2 MACH11 cluster topology using an RSS database server

7.1.4 High Availability Data Replication on an established environment

HDR defines a database server environment as always containing one primary and one secondary server in a pair. The secondary server acts as a stand-by server that can process automatically a failover in case of an outage. An HDR environment can be used as a database server provided mirroring solution. Both database servers maintain their own data sets in terms of dbspaces and chunks. All changes are replicated by shipping logical log files from the primary to the secondary server. These changes can be processed synchronously or asynchronously, depending on the configuration. HDR pairs can be easily integrated into heterogeneous networks containing different secondary server types.

7.1.5 Coexistence of the different server types in one MACH11 cluster

A MACH11 cluster can consist of a variety of secondary server types. You have many choices when it comes to combining the different server types in one environment. You may define multiple cluster environments according your organization’s requirements. A cluster can also be configured to exchange data with other clusters. One primary database server can be a part of multiple cluster environments.

Therefore, database servers can be treated as gateways between different organizations. You can also integrate the other replication features, such as ER and Flexible Grid, into the existing database server environments.

Figure 7-3 shows a sophisticated database server topology that represents a region acting locally but is also connected (by replicating data) to other regions. Both primary servers maintain the local cluster, but they also exchange data with the other cluster. In our example, we use ER for the replication. Using Flexible Grid would also work well for our environment.

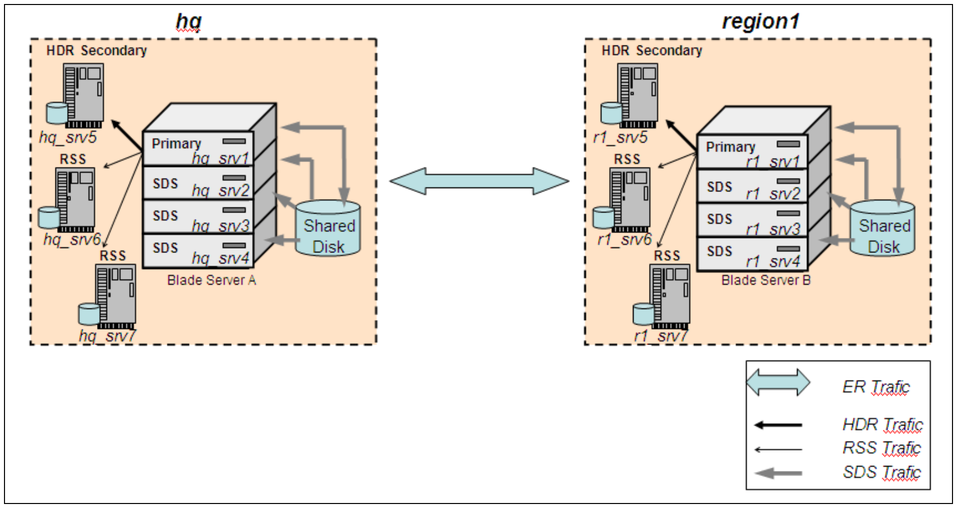


Figure 7-3 Integration of all MACH11 server together using replication

7.2 Common target database cluster topology

Let us now discuss how database cluster topologies can be used in a company's business environment. We introduce example database cluster topologies in detail and describe the different foci that they target.

7.2.1 Database cluster for ensuring availability

Availability is the prevention of a general outage of your data for a long time. Achieving availability is the first and major reason for setting up a database cluster. We begin with the introduction of a simple fallback solution, and grow the complexity to the global network of a database server infrastructure.

A local cluster that ensures hardware fault tolerance

The simplest goal in setting up a cluster is to ensure protection against hardware outages. This cluster can be set up locally in the same room. In this configuration, you might have one primary database server maintaining all the production data. There could be one to many standby servers on separate machines ensuring the reliability of the complete database server. Figure 7-4 show a possible layout of this local fault tolerance configuration.

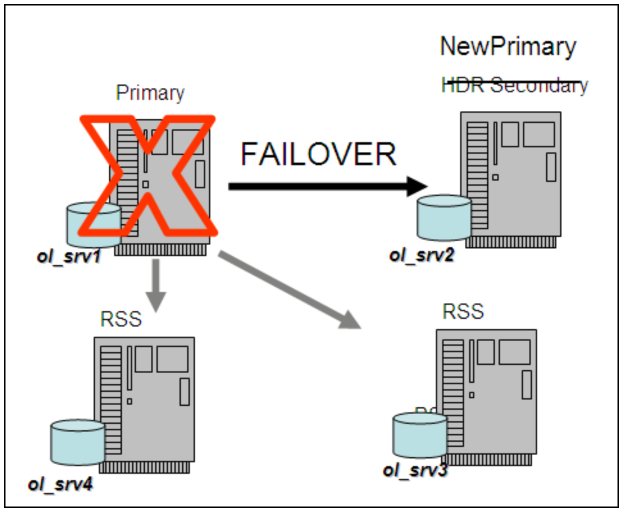


Figure 7-4 Ensuring data availability in case of a hardware outage

If you establish an automatic failover of the database server in combination with a redirection of the clients, the downtime in terms of database access is minimal. The example environment contains one primary server with two RSS servers attached to the primary and an HDR secondary server. In case of an outage, the switch of the primary role to the HDR secondary can be accomplished easily.

Achieving location independence

You can expand the simple example given Figure 7-4. If independence from a single database host machine is not enough, you can have multiple data centers connected together. The data centers themselves support load balancing and are clustered locally for hardware independence. In addition, we want to achieve location independence to survive catastrophic scenarios.

Figure 7-5 shows a possible cluster topology.

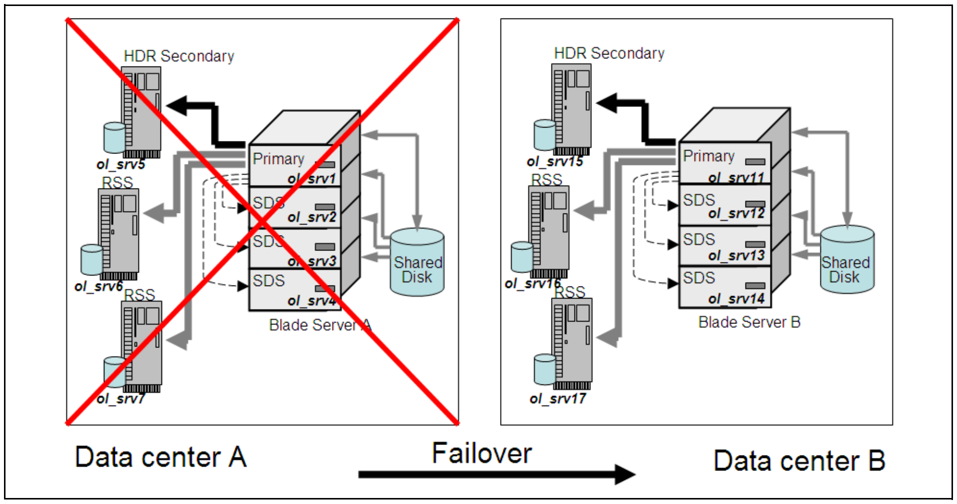


Figure 7-5 Ensuring location independence using a MACH11 cluster infrastructure

Securing your master database server

A common business scenario for data exchange in a company is to define a master database server. This particular database server takes control of the database schema of the production environment and the base data set, such as a master inventory or the customer's master data. Any changes to the data and schema or setting up a new database server are accomplished by using the master database server. You should have high availability on this type of master database server. You should have high availability on this type of database server to minimize downtimes and secure the business. The best solution would be to include this server in a cluster environment and secure the data with one or more standby database servers. The database server itself defines the one-to-many communication between the primary and all other servers that inherit the data. The complete data set is cloned to a standby server and possible changes are replicated. A failover policy needs to be applied in case of an outage.

Figure 7-6 shows a possible layout for this type of database server topology.

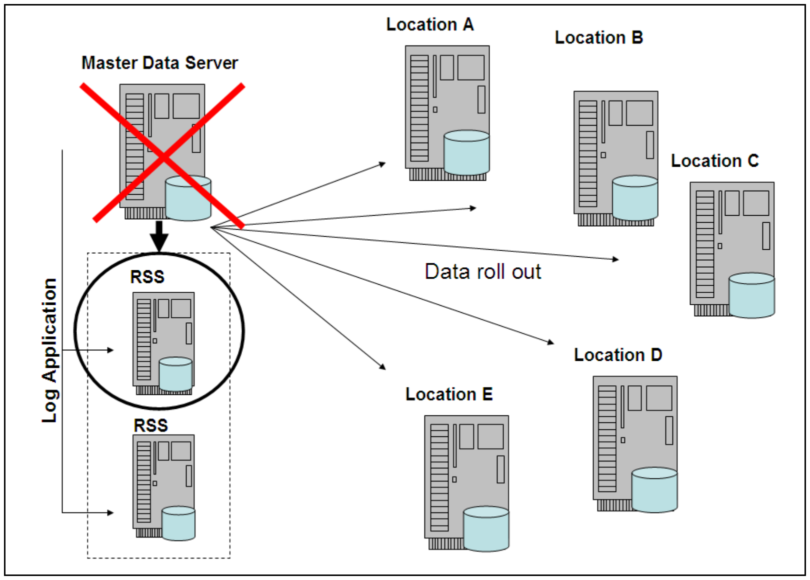


Figure 7-6   Secure the master database server in a one-to-many topology with a standby

7.2.2 Workload balancing

Data is maintained in the database server that provides access to the applications and the clients. Client applications are the source for generating, requesting, and maintaining the data stored in the database server. They are tied together. However, applications have different resource requirements in regards to the database server in terms of statement execution time expectations, memory usage, and CPU utilization. Using a set of database servers in a cluster that provides the same data consistently across the servers gives you the ability to specify categories of clients using different server resources.

When thinking about workload balancing, the application type is not the only item that should be considered; data access timing should also be considered. Applications do not use the database server evenly throughout the day, the month, or the quarter. During general business hours, applications often run in parallel. Off-peak hours are commonly used for scheduled batch data loading and aggregation. During business hours, the server is used for ad hoc queries, reporting, and heavy transaction processing in stores or subsidiaries.

Workload distribution

Imagine that your company defines a master data set maintained by a primary database server. There are various data accessing requests, such as a development department that requires production data for testing the new applications. You might have inexperienced users who are required to access the production data and their queries might produce Cartesian products or improper join criteria that returns huge result sets. You need to separate all these types of users to keep them from interfering with other database users. A possible solution is to replicate the master data to standby servers, which provides consistent data for access and actuality and can isolate user groups. With this approach, you provide access to all the users but avoid interruptions and disturbances in the production environment. Figure 7-7 shows a common cluster topology serving this type of client separation.

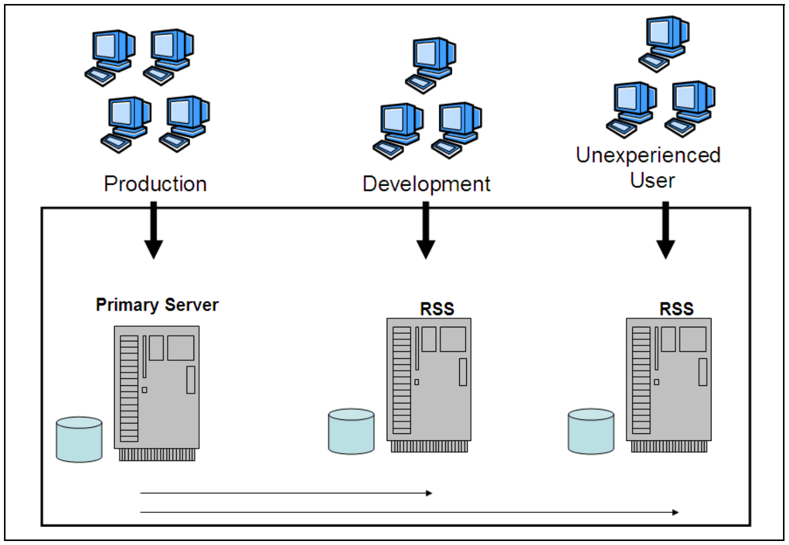


Figure 7-7 Workload distribution

Time-based workload balancing

Social networks are commonly used today. They keep members virtually connected through the Internet. The social network can be organized in one country or linked together across multiple regions. In this use case, let us look at a business day of a network of users in one country. During a full business day (24 hours), the data volume is lowest at night. The volume slightly increases at breakfast time as appointments for the day are made. Later on the there are people who use the social network at their business and children join the network after school. The peak usage is usually after working hours when everyone is socializing virtually.

The resource requirements for the database cluster fluctuates even for one day. The total cost of investment will be reduced if you have the flexibility to reduce the deployed resources at night time and increase the attached database servers during peak hours. Using the Informix SDS cluster solution, you can define your own cloud of database server resources so that you have flexibility regarding the number of servers; in addition, it is easy to switch them on and off.

Figure 7-8 shows a possible scenario for an database infrastructure. We define a flexible infrastructure for every day usage by attaching or detaching temporary database server resources that share the same subset of data. You can easily expand this example for monthly or quarterly usage.

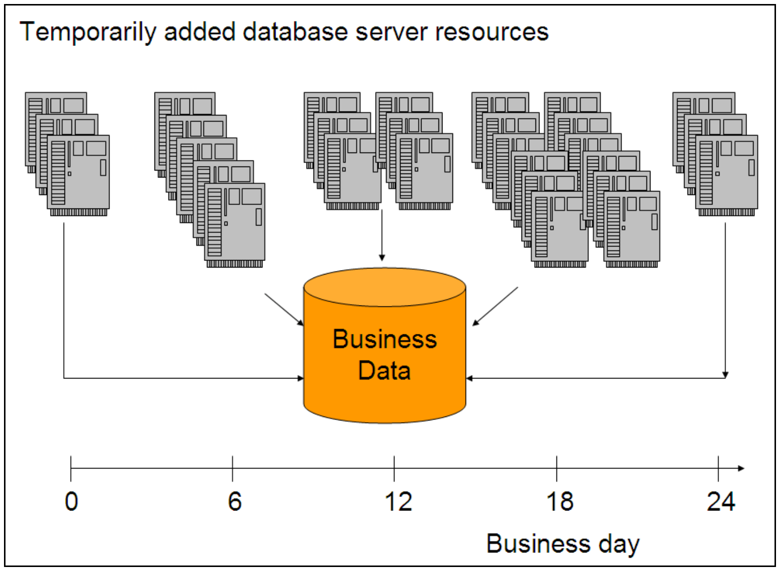


Figure 7-8 Define your own cloud for serving peak and off-peak application needs

7.3 General configuration guidelines

In the previous section, we discussed the strength of various Informix cluster solutions that you could use to design a topology that best reflects your organizational layout. After the topology is defined, the next step is to verify the potential changes that must be applied to the existing and the new database server infrastructures. You should ask yourself the following questions:

- Are there any additional resource requirements for the machine and storage used by the primary server?

- Do you need to change the general configuration of the existing database server when you set up a new cluster environment?
- What are the general requirements for the setup of the standby server in terms of ensuring the performance of the applications?

Here we discuss more detail the general resource requirements of the primary and the secondary database servers in the cluster with a focus on performance.

7.3.1 Categorizing server resource needs for MACH11 cluster

Database servers define their own resource requirements from the host machine. The major focus of the database server is on I/O throughput, CPU utilization, memory utilization, and network bandwidth. Using a server in a MACH11 cluster does not change the requirements in terms of these resources in comparison with a stand-alone server.

The secondary servers behave differently when applying the data replicated from the primary server. Thus, each secondary server type has its own requirements for providing the expected performance. Make sure the secondary servers are adequate for the workload. A major goal is to avoid falling behind in processing the data changes, as this lapse might block the primary server or interrupt the connection to the primary server. You have to focus on the database server owned resources, which must be adjusted when setting up an new or additional cluster server. Consider changing the following server maintained resources when planning a MACH11 cluster:

- Memory (primary database server): Logical log buffer size
- Disk (primary database server): Logical log file size
- Database schema (primary database server): Database logging mode
- Table definitions (fragmentation) index creation:
  - Affecting the size of logical log files
  - Affecting the workload balance on the recovery threads
- CPU utilization (secondary database server): Number of recovery threads

You might notice that most of these changes affect the primary server. Only the number of recovery threads must be applied on the secondary server. Let us discuss the parameters and their influence on the cluster behavior.

7.3.2 Roll-forward threads versus table and index creation and location

A MACH11 cluster ensures consistency of data across all participating database servers by replicating logical log files. All changes applied on the primary server are tracked in the logical log and periodically shipped to the secondaries. The secondary server retrieves the required information from the log and replicates the tracked changes to the local data set. Because this is continuous processing on the secondary server, the server stay in a continuous roll-forward mode. This roll-forward mode has some specific requirements.

Number of threads executing the work

The roll-forward of the log content for replicating on the secondary is executed by recovery threads that are organized at two levels. On the first level, there is one thread extracting the content from the log. This thread distributes each specified log activity for execution to one of the second level threads. The distribution of each operation is based on the unique table number, *partnumber*, on which the next log activity is based. The identified partnumber is hashed by the number of configured threads, which means activities on the same tables are replayed by the same thread. This action is required to keep the apply order specified by the log entries.

The amount of existing threads can be configured by setting the **`onconfig`** parameter OFF_RECVRY_THREADS. The default number of threads is 10. The guidelines for setting this parameter are as follows:

- Use a prime number.
- Use a minimum of 11 threads.
- Use, in general, more than three times of the number of CPU virtual processors.

Table and index definition guidelines

Changes on the same table are replayed on the secondary server by the same recovery thread. Frequently used tables, in view of changes, should be fragmented to parallelize the work on the secondary server to avoid too much lag behind on the secondary in comparison with the primary server.

For those tables with indexes, make sure that the index does not have a partition number that creates the same hash as the table itself.

Avoid keeping index and data in the same partition. Keeping index and data in the same partition generate more logs, especially update operation logs. Throttle the recovery thread if it has to apply other items in addition to the data.

7.3.3 CREATE INDEX statements that affect the logical log sizing

When a new index is created on a table on the primary server, it must be replicated. The layout of the index on the primary and the secondary servers must match because there are subsequent index operations, such as splits and compresses, that expect the same layout. The create index operations must be logged on the primary server on which RSS or HDR secondary servers are attached. This action is controlled by the LOG_INDEX_BUILDS configuration parameter. Set this parameter so the new index pages are also logged in the logical log and replicated. You should consider resizing the existing logical log size because of this additional requirement. Depending on the frequency of index builds and the size of the underlying table, additional log space is required to avoid long transactions.

7.3.4 Log buffer size and database logging modes

For performance reasons, the primary database server defines a set of memory buffers that are used as intermediate storage before the data is flushed to disk. Several factors affect the buffer flushing. If the database on the primary server is created with unbuffered logging or with ANSI mode, the content of the buffer is flushed to disk after a rollback or a commit is issued. In this case, the flush can happened anytime regardless of whether the buffer is full or not. Another criteria for a flush is that if one buffer becomes full, the flush is initiated, and the next buffer is used for the upcoming log entries.

When sizing the log buffer in a MACH11 cluster, it is a best practice to apply the following rules:

- Try to increase the logical log buffer size, which is defined by default as 64 KB.
- Try to avoid sending too many small log buffers from the primary server. The target server is expected to send an acknowledge after a certain time. If the message is not sent, the primary will either block or reset the connection.

In addition to these sizing guidelines, make sure that you can avoid creating databases with unbuffered logging. You will not be able to reduce the flush frequency by increasing the buffer size, as the buffer flush happens when a commit is issued.

7.4 The usage of the secondary servers

To achieve the most effective database application workload balancing, spread the clients based on their resource requirements across all the available database servers in the cluster environment. These servers include the secondary servers, regardless of whether the servers are updatable or not. You should identify the target database server that makes the best match between the application requirements and the functionality the server can provide.

In this section, we focus on the major differences in the behavior between a secondary server and a primary server. We discuss the limitations and their special setup requirements. We cover the following topics:

- Updatable secondary versus read-only standby servers: Supported SQL statements, including SELECT, data manipulation language (DML), and data definition language (DDL) SQL statements
- Concurrency support: Transactions, supported isolation levels, and lock behavior
- Temporary table use: Special configuration and setup requirements on secondary servers

7.4.1 Updatable secondary servers

In a MACH11 cluster, the default operation mode for a secondary server is read-only standby. The database server provides access for all incoming new sessions and allows the execution of any SELECT statements and SQL statements that will not change data. The only exception is the creation of non-logging temporary tables, because they use locally maintained temporary dbspaces.

The default behavior can be changed to enable the secondary database server to accept the Insert, Update, and Delete SQL statements either for executing within a transaction or as a single statement. The SQL statement itself is not processed on the local secondary server; it is redirected to the primary server. The primary server executes the requested operation and returns the status. By advancing the log position and the subsequent log replication, the changes are automatically transferred back to the requestor. The communication between the primary server and the secondary server is maintained, in an independent session, by internally started service threads called *proxy threads*.

To change the default behavior and support the execution of any Insert, Update, and Delete SQL statements on the secondary server, change the configuration parameter UPDATABLE_SECONDARY in the `onconfig` file. The default value is 0, which specifies that the server is operating as a read-only secondary server. To allow Insert, Update, and Delete SQL statements, change the parameter to a value higher than 0. The setting of this parameter influences the behavior of the secondary server in two ways:

- A setting other than zero enables the execution of the Insert, Update, and Delete statements. The setting defines how many threads are started to service the redirected writes.
- The parameter value has a performance impact on the local database server. The value defines how many service threads are started at server startup time for processing the requests redirected to the primary. Depending on the number of sessions and transactions executing the Insert, Update, and Delete statements in parallel, an appropriate number of service threads is required to handle all the scheduled requests in a timely manner. Do not specify more than two times of the number of the CPU virtual processors of the server.

Monitoring

The primary server and the secondary server exchange required information for a redirected DML operations. The parallel workload and the progress of the operations can be monitored by common interfaces, such as the sysmaster database and the onstat utility.

The sysmaster database

The sysmaster database provides the following tables for monitoring the operations:

- sysproxydistributors (on both database servers): Provides information about the number of proxy threads and their current statistics about the pending transactions.
- sysproxagents (on the primary database server): Provides statistics about sessions currently using updatable secondary functionality, such as assignments between the session and proxy thread, and the current error codes.
- sysproxysessions (on the secondary server): Provides the list of sessions and their assignment to the available proxy thread.

- sysproxytxns (on both database server): Provides more detailed information about the open transactions that handle the redirected writes and their associated proxy threads.
- sysproxtxnops (on the secondary): Provides more detailed information about the pending statements in an transaction, their base tables, and the current SQL status information.

The onstat utility

Using the onstat utility allows you to see a similar output on the command line level. The onstat options for monitoring are:

- **`onstat -g proxy`** (on both database servers)
  - This command provides the list of the available proxy threads on both database servers and, most importantly, the amount of pending operations in the transaction.
  - The output of this option matches the content of the sysproxydistributors table in the sysmaster database.
- **`onstat -g proxy all`** (on both database servers)
  - This option provides additional information about the sessions currently using the updatable secondary functionality. By combining this information with the output from the primary server, you can discover the thread pairs that are involved in the communication.
  - The output of this option matches with the content of the sysproxyagents table in the sysmaster database on the primary and the sysproxysessions on the secondary.

  Example 7-1 shows a sample output for this option.

Example 7-1 onstat -g proxy all on a updatable secondary

```
Primary    Proxy      Reference Transaction  Hot Row
Node       ID         Count     Count        Total
sds_prim   1530       10        0            0
sds_prim   1531       7         0            0
sds_prim   1532       7         0            0
sds_prim   1533       12        0            0
sds_prim   1534       13        1            0
sds_prim   1535       5         0            0
sds_prim   1536       8         0            0

Session  Session  Proxy    Proxy    Proxy    Current  Pending  Reference
         Ref      ID       TID      TxnID    Seq      Ops      Count
77       2        1536     152      1        4        1        1
```

- **`onstat -g proxy < proxy id >`** (only on the secondary)
  - Provides more detailed information about the current tasks on which the specified proxy thread is working.
  - The output of this option matches the content of the sysproxytxns table in the sysmaster database.
- **`onstat -g proxy <id> <txid>`**
  - Provides Information about the current processed statement and the base table. The SQL status information of the request is also shown.
  - The output of this option matches the content of the sysproxytxnops table in the sysmaster database.

  Example 7-2 shows a sample output of a particular proxy ID.

Example 7-2 onstat -g proxy <proxyid> and onstat -g proxy <proxid> <txid>

```
$ onstat -g proxy 1536
IBM IDS Version 11.70.FC2 -- Updatable (SDS) -- Up 02:53:33 -- 9657588 Kbytes
Proxy    Reference Pending  ProxySID
TxnID    Count     Ops
1        1         1        3

$onstat -g proxy 1536 1

Sequence Operation rowid    Table                         sqlerrno
Number   Type               Name
2        *ProxyWriteInsert 0       stores_demo:informix.tab      0
```

7.4.2 DDL statements

MACH11 secondary database servers do not support the execution of SQL based database schema changes (DDL statements) in sessions maintained locally. All DDL statements, such as creating new permanent tables, indexes, views, and triggers, have to be executed directly on the primary server of the cluster. After the statements process successfully, the changes are automatically replicated to the relevant secondary server.

If you want to have the flexibility to execute and replicate DDL statements within a cluster or a replication environment, consider the Flexible Grid functionality provided by the Informix database server. It is described in detail in Chapter 4, “IBM Informix Flexible Grid” on page 57.

7.4.3 Isolation levels and locking

Isolation levels and locks ensure the consistency of data for concurrent access on the local database server. Based on your definition for the secondary server in the MACH11 cluster environment, as a read-only or an updatable secondary server, the locking behavior and the supported isolation level are different.

Read-only secondary servers

The only isolation level that is supported in sessions maintained by a default read-only secondary server is *dirty read*. All the local sessions always receive the latest version of the replicated data, regardless if the last change was already committed or not.

On a read-only standby secondary database server, the replicated logs are applied by the fast recovery threads in a continuous roll-forward mode. Any changes on the data, whether they are triggered by single or multiple statement based transactions, are not locked during the lifetime of the transaction on this secondary server type. The reason is simple. The secondary server is an exact copy of the primary. On the primary, only the successful data changes are logged. Attempted operations on data that failed, for example, because of locking conflicts, are not applied to the logical log on the primary server and thus are not replicated. Therefore, the target server can taken it for granted that no conflicts can happen when applying the data. The recovery threads only need to ensure the order of the changes according to the primary server. Because the secondary itself behaves as read-only, there are no locking conflicts to be expected between local sessions and roll-forward threads.

Updatable secondary servers

The updatable secondary server supports the dirty read and the committed read isolation levels. The default isolation level for all incoming new sessions on the secondary is dirty read.

In addition to the dirty read and committed read isolation levels, the uselastcommitted feature also can be used. Uselastcomitted enables the reader of the data to obtain the before image of a pending change processed by any parallel open transaction. You can either set the USELASTCOMMITTED **onconfig** parameter to your preferred isolation level, or you can dynamically set the uselastcommitted using the following SQL statements in you current session:

```
set environment uselastcommitted “Dirty Read”;
set environment uselastcommitted “Comitted Read”;
set environment uselastcommitted “All”;
```

To use the uselastcomitted functionality, the table from where you read the data must support row level locking. It does not work with page level locking.

The cursor stability and repeatable read isolation levels are not supported on updatable secondary servers.

Because the sessions are allowed to apply changes in parallel, for any open transaction, the updatable secondary server must know about the locks. The secondary server does not send the lock request to the primary for checking possible lock conflicts; it can be done locally. To ensure that a lock conflict situation is detected correctly, the updatable secondary server creates the locks locally for all changes applied by the logical log roll-forwarding. The owner of all the locks are the recovery threads. The locks are removed when the commit or rollback of the open transactions is indicated by a certain entry in the logical log. Also, any changes triggered by the local session are redirected to the primary server and are applied there. Because of advancing the log position and the replication of the log from the primary back to the local server, the lock is created on the secondary server as well.

7.4.4 Temporary table use

The secondary server supports the usage of temporary tables in general. The temporary tables can be created in local database server sessions by using the SQL statement CREATE TEMP TABLE or SELECT INTO TEMP. The temporary tables created by the database server for view materialization, sorting, and hash table overflow are also supported on the secondary server. The only requirement is that the temporary table has to be created without logging.

To make sure that all temporary tables are created in temporary dbspaces, set the TEMPTAB_NOLOG parameter in the `onconfig file` to 1 in the MACH11 secondary server. By using this setting, the secondary server forces the creation of all the temporary tables without logging.

To support the temporary table, the secondary server maintains its own subset of temporary dbspaces. Depending on the server types, the definition of the temporary dbspaces can differ. Using HDR or RSS cluster environments, the primary database server define the location and the size of the chunks assigned to the dbspaces. On SDS cluster environments, to ensure a unique location, each secondary database server defines its own set of temporary dbspaces within the local configuration file.

7.5 Failover

Maintaining the data consistently in a cluster environment that contains database servers with different responsibilities and roles is one part of the functionality of a database cluster solution. The other major function is managing failover, manually or automatically, depending on your requirements. The goal of failover is to switch responsibilities between servers when a database server faces a planned or unplanned outage situation and becomes unavailable. The failover process contains both the database server role switch and the redirection of the clients. The failover process can be automatic or manual.

The automatic failover and the database client redirection is, most of the time, provided by the Connection Manager process. The Connection Manager is a intermediate level of communication between an client application and the database server. It knows about the defined resources in the cluster and their current status in terms of utilization and availability. The process itself maintains the definition of the client distributions across the available resources granted to a client group. The definition contains the database server granted to the group and defines priorities. Clients are redirected based on the distribution priorities in case one of the database server becomes unavailable.

The Connection Manager also defines a failover strategy for the database server based on the predefined rules. The strategy is automatically applied to the cluster in case the primary database has an outage. Checking the availability of the managed database resources is a major task of the Connection Manager, because it has to distribute the clients based on the workload and availability. For more details about the Connection Manager and failover process, see Chapter 12, “Informix in cloud and virtualization environments” on page 461.

The failover process can also be applied manually. It can be accomplished by running the **onmode** command. An emergency is not the only case where the failover of the database server is required. Manual failover also can be used in other situations, such as server updates, planned hardware maintenance, and replacement activities. In the following sections, we give a detailed overview of server failover based on sample cluster environments, including how a manual change of the server type can be applied. We describe what has to be considered before the server type is changed. We start our discussion with the takeover using sample simple cluster layouts. We also show the movement of roles in a cluster environment containing different database server types.

7.5.1 Manual failover in a unique SDS cluster environment

Figure 7-9 shows a sample SDS cluster layout that maintains only an SDS database server. The cluster defines a primary database server with two secondary servers attached. All three servers are connected to a SAN where the dbspaces set is maintained, so the storage is on a host machine independent of the database server.

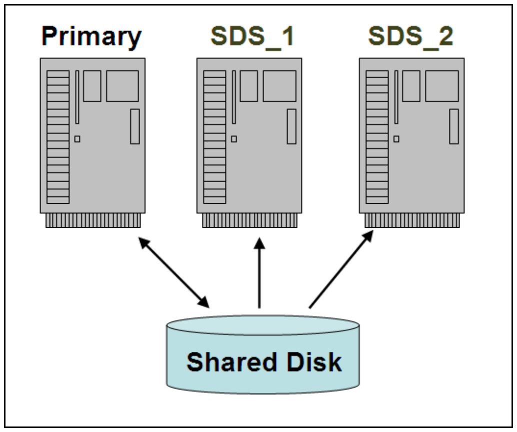


Figure 7-9 Shared Disk Secondary servers

Planned server outage

If the primary server has to be taken out of service for a planned outage, the primary server role must be transferred to either of the two SDS servers. Because both SDS servers are reading the same disk subsystem, it does not matter which of the two SDS servers becomes the primary server; they are equally able to assume the role of the primary server if they have a similar size.

How does the transfer work? In an SDS cluster environment, the new primary server notifies the old primary server that it is taking over the ownership of the shared disk. The old primary server then rolls back any open transactions and notifies the new primary server that the rollback is complete. After this process is complete, the old primary server shuts down. This action is the signal for the new primary server to continue. Promoting an SDS server to the primary role is performed by issuing an **onmode** command:

```
$ onmode -d make primary sds_1
```

Unplanned outage: Primary database server outage

Imagine that the primary database server becomes unavailable because of an outage, such as a hardware or network outage. In this situation, the connection between the new and old primary server does not exist. You have to use the force option of the **onmode** command to complete the primary role change.

Issue the **onmode** command with the force option to initiate the manual failover from an SDS primary to an SDS secondary server, as shown here:

```
$ onmode -d make primary SDS_1 force
```

The **onmode** command force option should only be issued when it is certain that the original primary server is not active. Because the force keyword cause the new primary server to become the source server without communicating with the old primary server, it can cause database corruption if the old primary server is still active.

Final cluster layout

Figure 7-10 shows the possible cluster layouts after processing the manual failover. The new roles are independent of the switch triggered by the planned or unplanned outage.

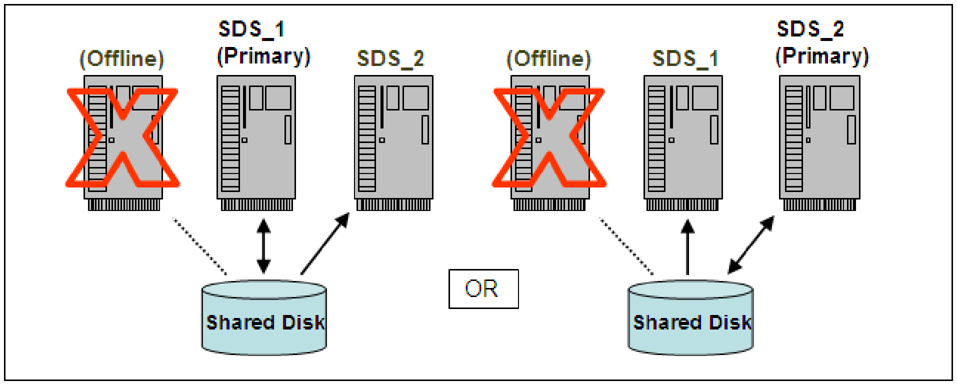


Figure 7-10 Transfer of the primary role to one of the SDS servers

If your Informix database cluster infrastructure includes HDR Secondary and RSS, then the new primary notifies all existing secondary servers about the switch of a primary server. All secondary servers have to reconnect to the new primary for a new synchronization of the log replication.

At some later time, the old primary server can be brought up as an SDS server simply by issuing the **`oninit`** command, provided that the SDS_TEMPDBS and SDS_PAGING parameters are set correctly in the `onconfig` file before issuing the **`oninit`** command. The role of this SDS server can then be changed back to the primary using the make primary option of the **`onmode`** command.

7.5.2 Mixing HDR and RSS

Here we discuss more complicate environments with different types of secondary servers. First, we introduce the sample cluster environment and how the server can be established. After the setup is complete, we continue discussing possible failover scenarios in the sample cluster.

RSS servers can coexist with an HDR pair, as shown in Figure 7-11. Note that only one HDR secondary server can be connected to a primary along with any number of RSS servers.

This kind of mixed environment can be useful in many cases, for example:

- Additional availability for disaster recovery and for remote reporting
- Additional remote backup

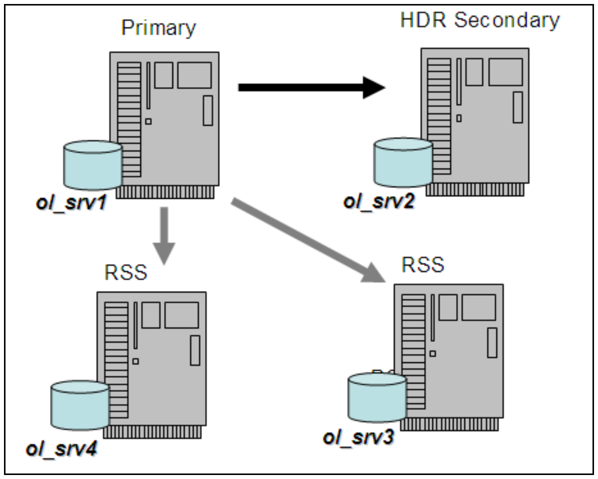


Figure 7-11 HDR and RSS coexistence

Table 7-1 shows the sample steps to be executed at each server to set up the configuration shown in Figure 7-11 on page 331.

Table 7-1 Steps to set up a typical RSS and HDR configuration

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Bring up primary.	`oninit`			
Take Level 0 backup at primary.	`onbar -b -L 0`			
Register HDR secondary server.	`onmode -d primary ol_srv2`			
Register RSS server.	`onmode -d add RSS ol_srv3`			
Register RSS server.	`onmode -d add RSS ol_srv4`			
Do physical restore the Level 0 backup at secondaries.		`onbar -r -p`	`onbar -r -p`	`onbar -r -p`
Connect secondaries to primary.		`onmode -d secondary ol_srv1`	`onmode -d RSS ol_srv1`	`onmode -d RSS ol_srv1`

In the following sections, we discuss various scenarios that are handled while servers are configured as a mix of HDR and RSS setups.

Failover of a primary to an HDR secondary

Suppose the primary server is scheduled for a planned shutdown, such as for maintenance, or an unplanned shutdown, such as when power is lost or there is a disaster situation. In these situations, the HDR secondary can be promoted to assume the role of primary, as shown in Figure 7-12. Here, all the RSS nodes are connected to the new primary server.

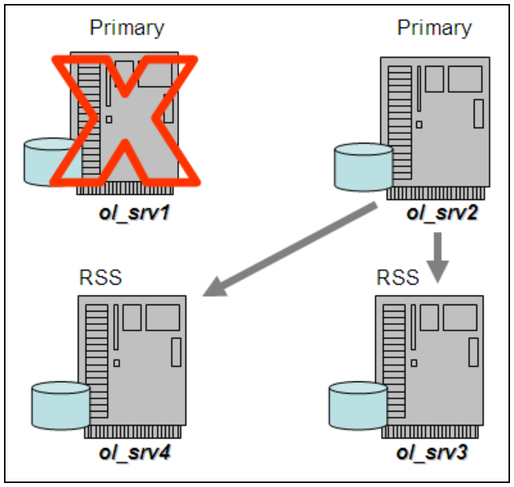


Figure 7-12   Failover of a primary to a secondary

Table 7-2 lists the commands and sequence for this situation.

Table 7-2   Sample steps to fail over a primary to an HDR secondary

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Old primary is down.	Down			
Convert HDR secondary to primary.		`onmode -d make primary`		
RSS nodes connect to new primary.			Connected to ol_srv2	Connected to ol_srv2

Swapping roles of the HDR primary and HDR secondary

To swap the roles of the HDR primary and HDR secondary, use the steps given in Table 7-3. The RSS servers connect to the new primary after swapping.

Table 7-3 Sample steps to swap the HDR primary and HDR secondary roles

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Convert HDR secondary to HDR primary.		`onmode -d make primary`		
Old primary stopped.	Server stopped			
RSS nodes connect to new primary.			Connected to ol_srv2	Connected to ol_srv2
Bring up old primary as HDR secondary.	`oninit -PHY`			

The result of swapping roles is shown in Figure 7-13.

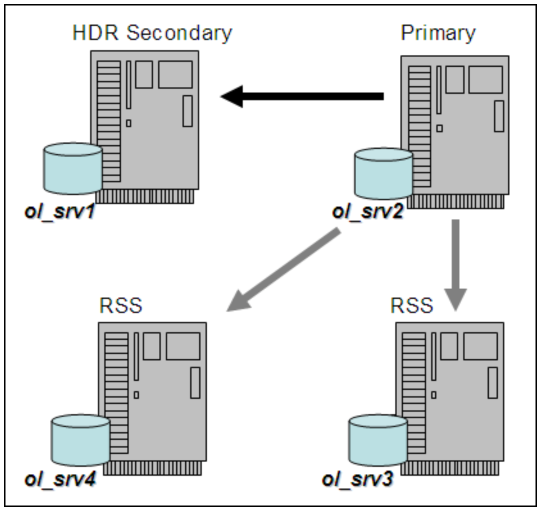


Figure 7-13 Swapping the HDR primary and HDR secondary roles

Failover of the HDR secondary to an RSS node

Suppose in the initial setup that the HDR secondary is taken out of service or destroyed. Instead of setting up a new HDR secondary, you can immediately promote one of the RSS nodes to an HDR secondary, as shown in Figure 7-14.

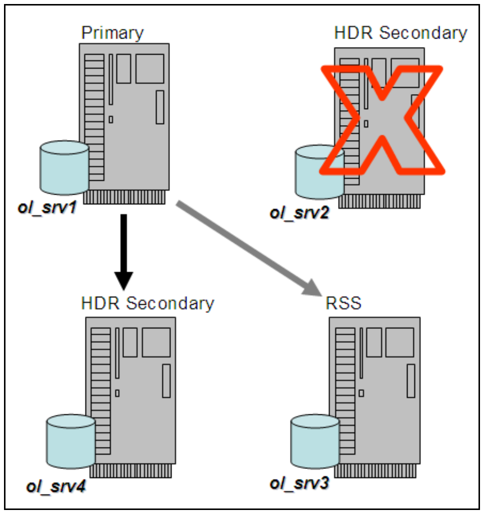


Figure 7-14 Failover of an HDR secondary to RSS

The sample steps are given in Table 7-4.

Table 7-4 Sample steps to fail over an HDR secondary to an RSS node

Task	**ol_srv1**	**ol_srv2**	**ol_srv3**	**ol_srv4**
HDR secondary is out of service or destroyed.		Down		
Convert ol_srv4 to HDR secondary.				`onmode -d secondary ol_srv1`

Note that to bring the old HDR secondary back, there are two options:

- Convert the new HDR secondary back to RSS, then start the old HDR secondary.
- Start the old HDR secondary as an RSS node.

Promoting an HDR secondary to RSS

To convert an HDR secondary to an RSS secondary so that all the nodes are RSS (from the layout shown in Figure 7-11 on page 331 to the layout shown in Figure 7-15), use the commands given in Table 7-5.

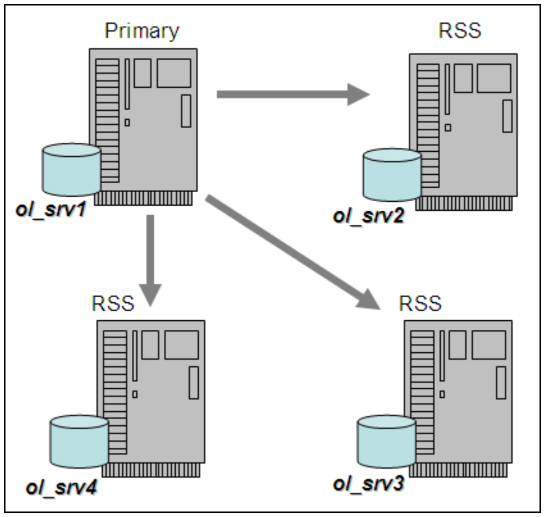


Figure 7-15 Promoting an HDR secondary to RSS

Table 7-5 Sample steps to promote an HDR secondary to RSS

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Register HDR secondary as an RSS node at the primary server.	`onmode -d add RSS ol_srv2`			
Convert HDR secondary to RSS.		`onmode -d RSS ol_srv1`		

Making the HDR a standard server

To make the HDR secondary server in the initial setup to be a standard server (Figure 7-16), break the HDR connection between the primary server and HDR secondary server.

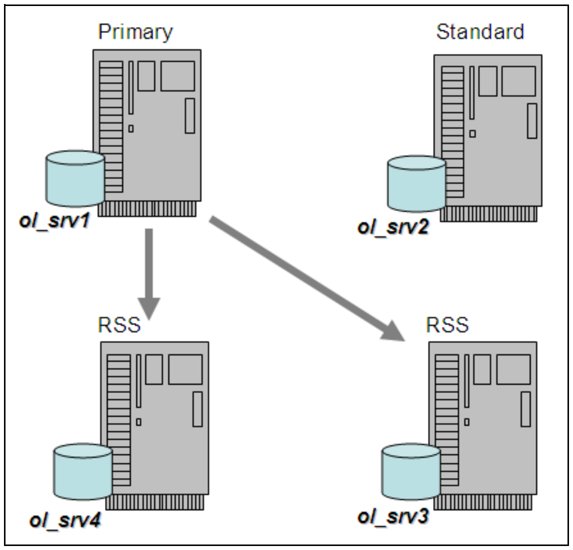


Figure 7-16 Breaking the HDR connection at the HDR secondary

Table 7-6 gives the commands to break an HDR connection from the primary server.

Table 7-6 Sample command to break an HDR connection from the primary server

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Make the HDR secondary a stand-alone server.		`onmode -d standard`		
	Connection to ol_srv2 dropped			

Making an RSS node into a standard server

To convert an RSS node to a stand-alone server, there are two options:

- Execute **onmode -d** to make the primary an RSS. This command shuts down all the other peer nodes, as shown in Figure 7-17.

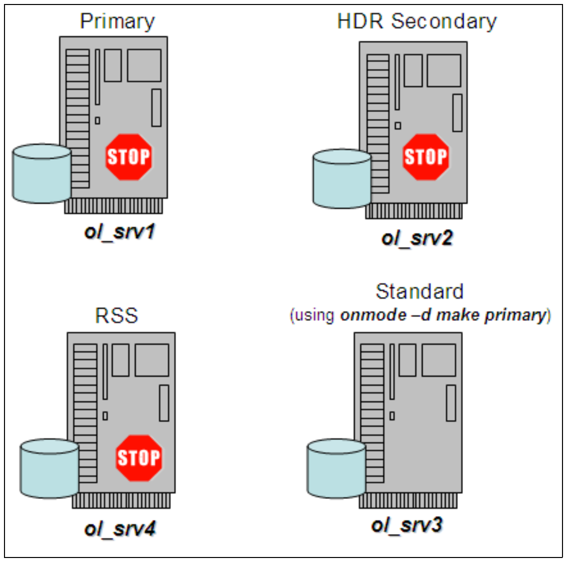


Figure 7-17 Making an RSS node a standard server (Option 1)

The commands for this option are given in Table 7-7.

Table 7-7 Sample commands to convert RSS to a stand-alone server (Option1)

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Convert ol_srv3 node to a stand-alone server.			`onmode -d make primary`	
	Server stopped	Server stopped		Server stopped

- Execute **onmode -d standard** at the RSS node and the result will be as shown in Figure 7-18.

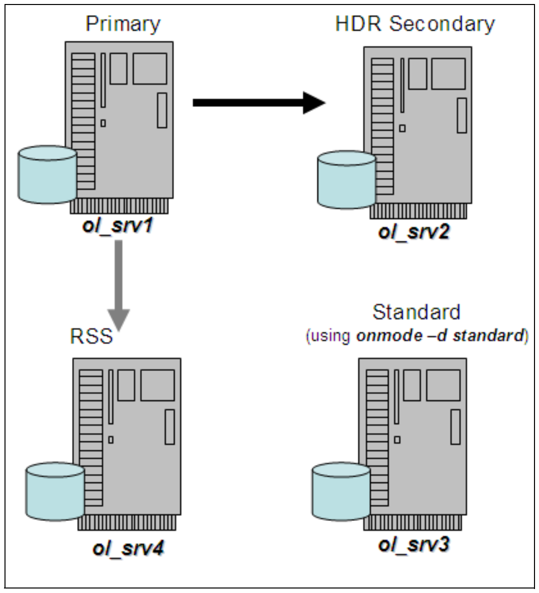


Figure 7-18 Making an RSS node a standard server (Option 2)

The commands for this option are shown in Table 7-8.

Table 7-8 Sample commands to convert RSS into a stand-alone server (Option2)

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Convert ol_srv3 to a stand-alone server.			**onmode -d standard**	
	Connection to ol_srv3 dropped			

Breaking the HDR connection at the primary

The HDR connection between the primary and HDR secondary can also be broken at the primary using the commands lists in Table 7-9. Note that the RSS servers still remain connected to the primary server.

Table 7-9 Sample commands to break the HDR connection at the primary server

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4
Break HDR connection at the primary server.	`onmode -d standard`			
		Drop connection to ol_srv1.		

Figure 7-19 shows the result of this action.

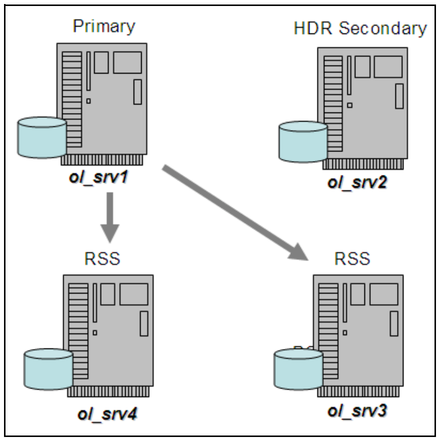


Figure 7-19 Breaking the HDR connection at the primary server

7.5.3 Mixing HDR and SDS

In this section, we discuss the cluster containing HDR and SDS database servers. We describe the options for possible switches.

SDS servers can coexist with an HDR pair, as shown in Figure 7-20. Note that only one HDR secondary can be connected to a primary along with any number of SDS servers.

An useful scenario for this type of mix is adding additional server nodes as on demand servers for handling a sudden increase of read-only clients along with a remote failover capability.

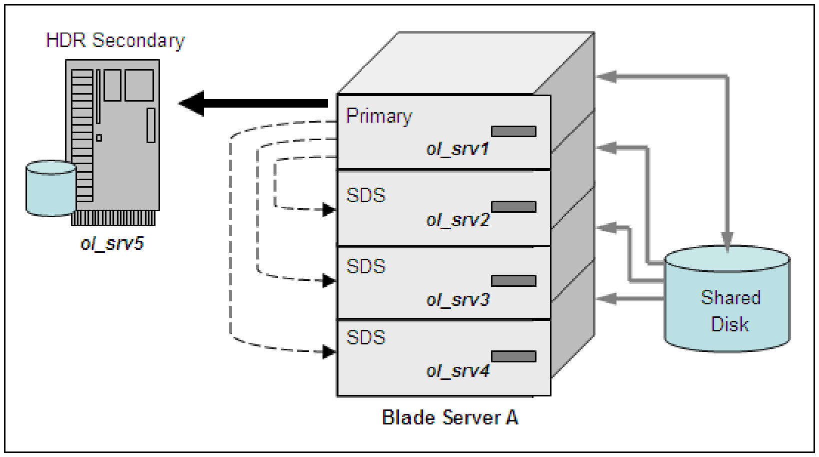


Figure 7-20 HDR and SDS coexistence

Table 7-10 shows the sample steps to configure this setup.

Table 7-10 Sample steps to configure servers with HDR and SDS

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Start server ol_srv1.	`oninit`				
Register ol_srv1 as an SDS primary server.	`onmode -d set SDS primary ol_srv1`				
Start all SDS nodes.		`oninit`	`oninit`	`oninit`	
Take a level 0 backup at the primary.	`onbar -b -L 0`				
Register ol_srv1 as an HDR primary.	`onmode -d primary ol_srv5`				

Task	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Restore at ol_srv5.					`onbar -r-p`
Register ol_srv5 as an HDR secondary and establish HDR with the primary server.					`onmode -d secondary ol_srv1`

Failover of a primary to an HDR secondary

When the primary goes down, the HDR secondary is made to the HDR primary; however, all the SDS nodes go down, as shown in Figure 7-21.

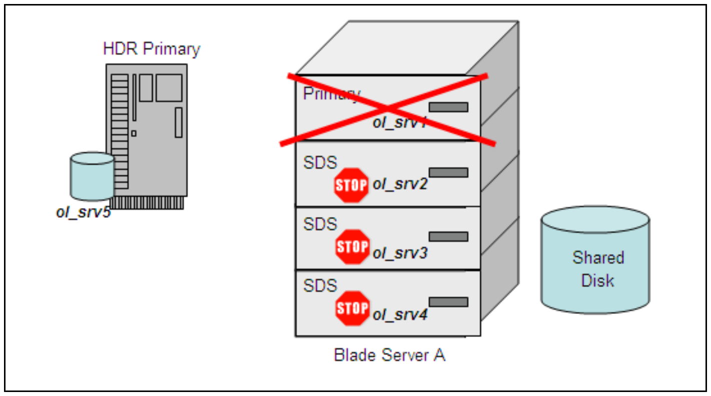


Figure 7-21 Failover of primary server to HDR secondary

Table 7-11 lists the sample steps to accomplish this task.

Table 7-11 Sample steps to fail over a primary to an HDR secondary

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
The primary is down.	Down				
Convert the HDR secondary to a primary.					`onmode -d make primary`
All SDS nodes connected to the old primary will be shut down.		Shut down server.	Shut down server.	Shut down server.	

Swapping the role of the HDR primary and HDR secondary

An variation of failover is swapping the role of the HDR primary server and the secondary server. In this scenario, although both HDR primary and secondary servers are up after swapping, all the SDS secondary servers are down.

Table 7-12 lists the sample steps to swap the roles of the HDR primary and HDR secondary, along with the status of the SDS servers after the swap. When the HDR secondary is made the primary, all SDS nodes go down if the **`onmode -d make primary`** command is used.

Table 7-12 Sample steps to swap the HDR primary and HDR secondary.

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Make the HDR secondary the new primary.					**`onmode -d make primary`**
All the peer nodes are shut down.	Server shutdown	Server shutdown	Server shutdown	Server shutdown	
Do a physical recovery at the old primary.	**`oninit -PHY`**				
Convert the old primary as an HDR secondary.	**`onmode -d secondary ol_srv5`**				

If the traditional method for swapping is used, the SDS nodes go down when the old primary is brought up in physical recovery mode, as shown in Figure 7-22.

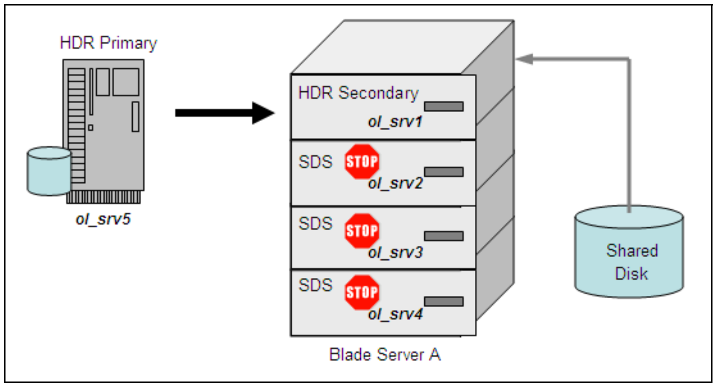


Figure 7-22 Swapping the roles of the HDR primary and HDR secondary

Table 7-13 lists the sample steps to accomplish this task.

Table 7-13 Traditional swapping of the HDR primary and secondary

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Bring down the primary.	`onmode -ky`				
Convert the HDR secondary to an HDR primary.					`hdrmkpri.sh`
Convert the primary to HDR secondary.	`hdrmksec.sh`				
All SDS nodes connected to ol_srv1 must be shut down.		Server shutdown	Server shutdown	Server shutdown	
Bring up the new primary.					`oninit`

7.5.4 Failover of the primary to an SDS node

If the primary server is down, one of the SDS nodes can be made the primary. In this case, the HDR secondary connects to the new primary and the other SDS nodes re-connect to the new primary, as shown in Figure 7-23.

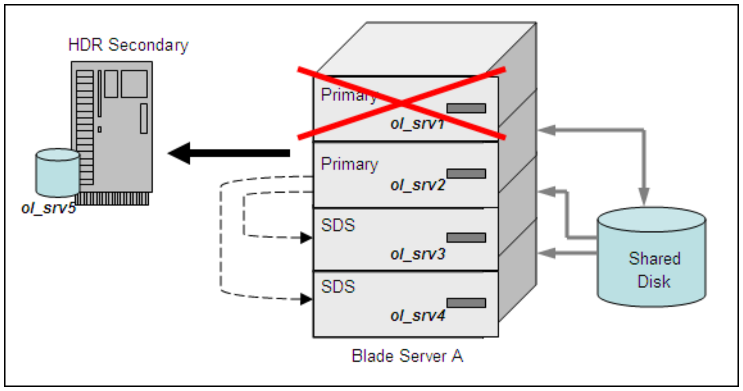


Figure 7-23 Failover of primary to an SDS node

Table 7-14 lists the sample steps to accomplish this task.

Table 7-14 Sample steps to fail over a primary to an SDS node

	ol_srv1	**ol_srv2**	**ol_srv3**	**ol_srv4**	**ol_srv5**
	Down				
Convert the SDS node to a primary node.		`onmode -d make primary -force ol_srv2`			
All other SDS nodes will connect to the new primary.			Connected to ol_srv2	Connected to ol_srv2	
The HDR secondary server re-establishes connection with the new primary.					Connect to the new HDR primary ol_srv2.

Swapping the primary to an SDS node

The role of the primary can be swapped to an SDS node by converting one of the existing SDS nodes to assume the primary role. After this swap, the HDR secondary and other SDS servers connect to the new primary, as shown in Figure 7-24.

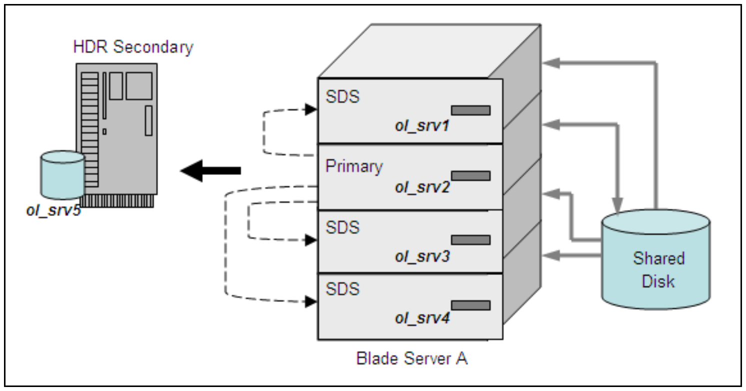


Figure 7-24 Swap the primary to an SDS role

Table 7-15 lists the sample steps to accomplish this task.

Table 7-15 Sample steps to convert a primary to an SDS node

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Make the ol-srv2 SDS node the primary.		`onmode -d make primary ol_srv2`			
The old primary shuts down.	Shut down the server.				
The other SDS nodes connect to the new primary.			Connected to ol_srv2	Connected to ol_srv2	
The HDR secondary re-establishes the HDR connection with the new primary.					Connect to the new HDR primary ol_srv2.

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Bring up the old primary as an SDS node.	`oninit`				
ol_srv1 connects to the new primary as an SDS node.	Connect to ol_srv2.				

Converting the HDR secondary to a stand-alone server

In the initial setup (Figure 7-20 on page 341), you can convert the HDR secondary server to be a stand-alone server by breaking the HDR link either at the HDR secondary or from the primary. The SDS nodes continue to work because they are connected to the primary. Figure 7-25 illustrates the layout after the HDR connection is broken.

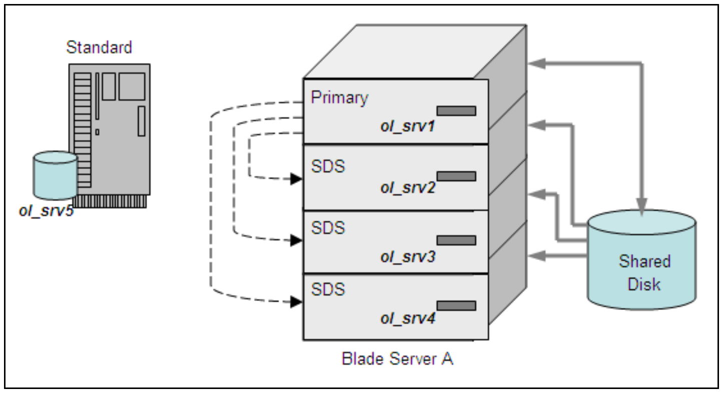


Figure 7-25 Breaking the HDR connection at the HDR secondary

Table 7-16 shows how to accomplish this task.

Table 7-16 Sample steps to break the HDR connection at the HDR secondary

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Convert the HDR secondary to standard.					`onmode -d standard`
	The HDR connection to ol_srv5 is dropped.				The HDR connection to ol_srv1 is dropped.
		Still connected to ol_srv1	Still connected to ol_srv1	Still connected to ol_srv1	

Breaking the HDR connection at the primary

Here we show the effect of breaking the HDR connection at the primary. Note that the SDS nodes and their connection to the primary remain unaffected, as shown in Figure 7-26.

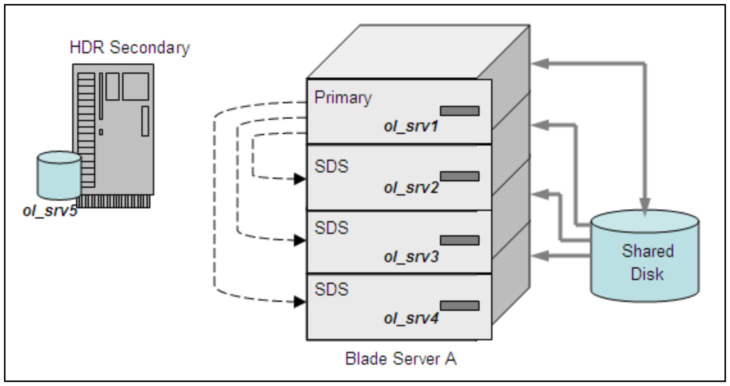


Figure 7-26 Breaking the HDR connection at the primary

Table 7-17 lists the sample steps to accomplish this task.

Table 7-17 Sample steps showing the effect of breaking HDR at the primary

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5
Break the HDR connection at the primary.	`onmode -d standard`				
The HDR connection is broken.	Disconnected from HDR secondary				Disconnected from primary
SDS nodes are still connected to the primary.		Still connected to ol_srv1	Still connected to ol_srv1	Still connected to ol_srv1	

7.5.5 Mixing RSS and SDS

SDS servers can coexist with RSS nodes, as shown in Figure 7-27. In this setup, one primary server connects to all the SDS servers and the RSS nodes.

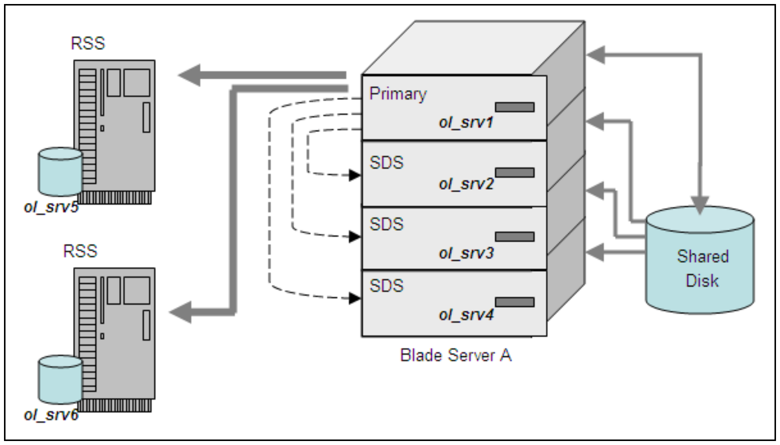


Figure 7-27 RSS and SDS coexistence

This type of mixture can be useful in the following scenarios, as well as others:

- On demand addition of server nodes for handling a sudden increase of read-only clients along with multiple remote read-only servers.
- RSS servers can be used as the remote backup servers, so that in the event of any catastrophe at the location where the primary SDS combination is located, the RSS servers can be converted to standard servers and make the organization's data available to the clients.

Table 7-18 lists the steps to set up this configuration.

Table 7-18 Sample steps to set up RSS and SDS nodes together

	ol_srv1	ol_srv2	ol_srv3	ol_srv4	ol_srv5	ol_srv6
Start ol_srv1.	`oninit`					
Register ol_srv5 as RSS.	`onmode -d add RSS ol_srv5`					
Register ol_srv6 as RSS.	`onmode -d add RSS ol_srv6`					
Register ol_srv1 as the SDS primary.	`onmode -d set SDS primary ol_srv1`					
Make a Level 0 backup.	`onbar -b -L 0`					
Start the SDS nodes.		`oninit`	`oninit`	`oninit`		
Restore level 0 at the RSS nodes.					`onbar -r -p`	`onbar -r -p`
Establish the RSS connection with the primary.					`onmode -d RSS ol_srv1`	`onmode -d RSS ol_srv1`

Failover of the primary to an SDS node

After the primary fails over to one of the SDS nodes, all the RSS nodes drop their connection to the old primary and re-establish their connection to the new primary server. All the other SDS nodes also connect to the new primary server, as shown in Figure 7-28.

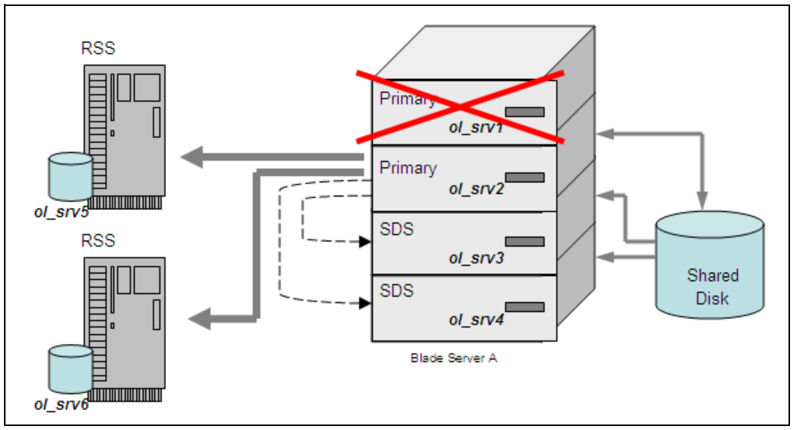


Figure 7-28 Failover of the primary to an SDS node

Transferring the primary to an SDS node

After the SDS server becomes a new primary and the old primary is swapped to an SDS, all the RSS nodes drop their connections to the old primary and re-establish their connection to the new primary server. All other SDS nodes also connect to the new primary server, as shown in Figure 7-29.

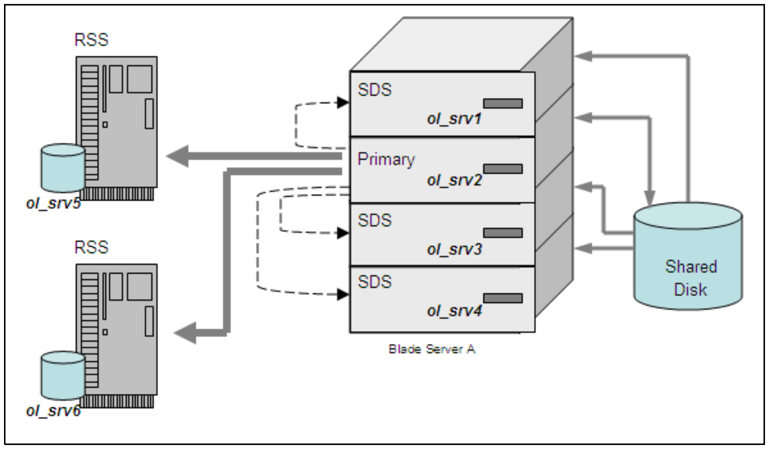


Figure 7-29 Transfer of a primary into an SDS node

Making an RSS node a standard node

An RSS node can be made into a stand-alone server, and there are two options for completing that task. Select the option based on your business needs:

- Use the **`onmode -d make primary`** command at the RSS node. This actions stops all the peer nodes, as shown in Figure 7-30.

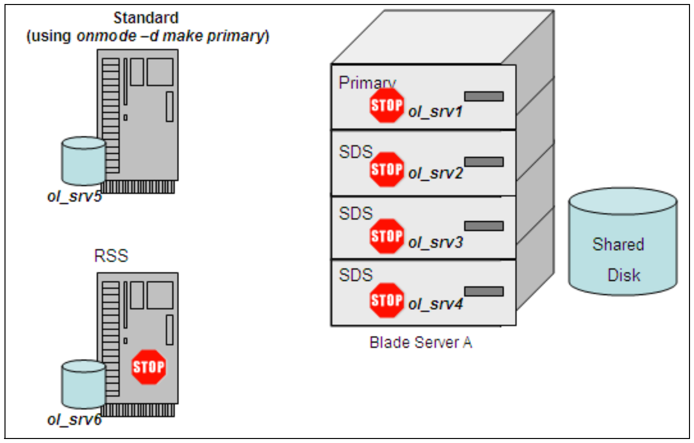


Figure 7-30 Converting an RSS server to a stand-alone server

- Use the **`onmode -d standard`** command if you just want to take one of the RSS servers out of the cluster and need to use it as a stand-alone server. The remaining cluster nodes stay intact. This scenario is shown in Figure 7-31.

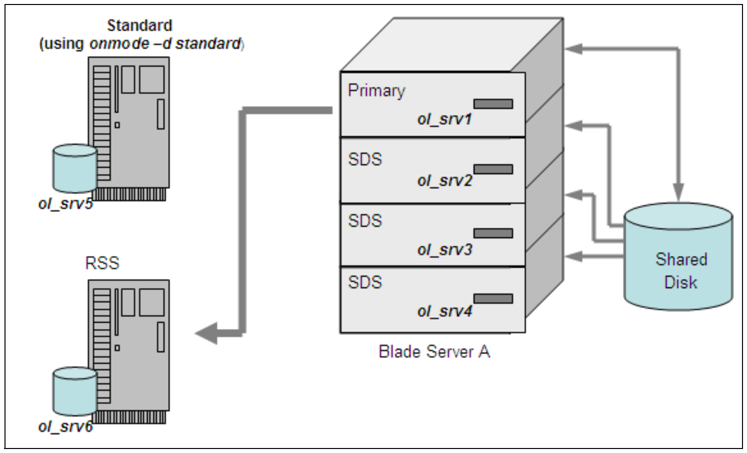


Figure 7-31 Converting RSS to a stand-alone server using onmode -d standard

Note that after the RSS node is taken out of the cluster and converted to a standard server, then there is no more data flowing between the primary server of the cluster to this server. To put this node back into the cluster, you have to make a Level 0 backup of the primary and restore it, and re-register this server as an RSS node.

7.5.6 Using the onmode command in an HA configuration

We have discussed the use of the **`onmode -d make primary`** command throughout the previous sections. We summarize the effect of the **`onmode -d`** command in making the primary in an HA configuration in Table 7-19.

Table 7-19 Effect of running onmode -d to make a primary in an HA environment

New primary	Peer node	Action
SDS Node	SDS	Attach to the new primary and continue.
	RSS	Attach to the new primary and continue.
	HDR secondary	Attach to the new primary and continue.
	Old primary	Shut down.
HDR secondary	SDS	Shut down.
	RSS	Attach to the new primary and continue.
	HDR primary	Depends on user action.
RSS Node	SDS	Shut down.
	HDR	Shut down.
	RSS	Shut down.

7.6 Administration and monitoring

The following user interfaces are available to administer and monitor the MACH11T cluster:

- The OpenAdmin Tool (OAT) for web-browser based administration.
- The **`onstat`** command for command-line monitoring.

7.6.1 OpenAdmin Tool

The OAT provides the following features for an Informix cluster:

- Informix cluster detection and layout retrieval
- Status monitoring of the cluster member in terms of availability and workload
- Changing the database server mode in the cluster in terms of startup and shutdown
- Adding a new SDS to an existing cluster environment
- A Connection Manager service level agreement (SLA) and failover arbitrator strategy definition maintenance

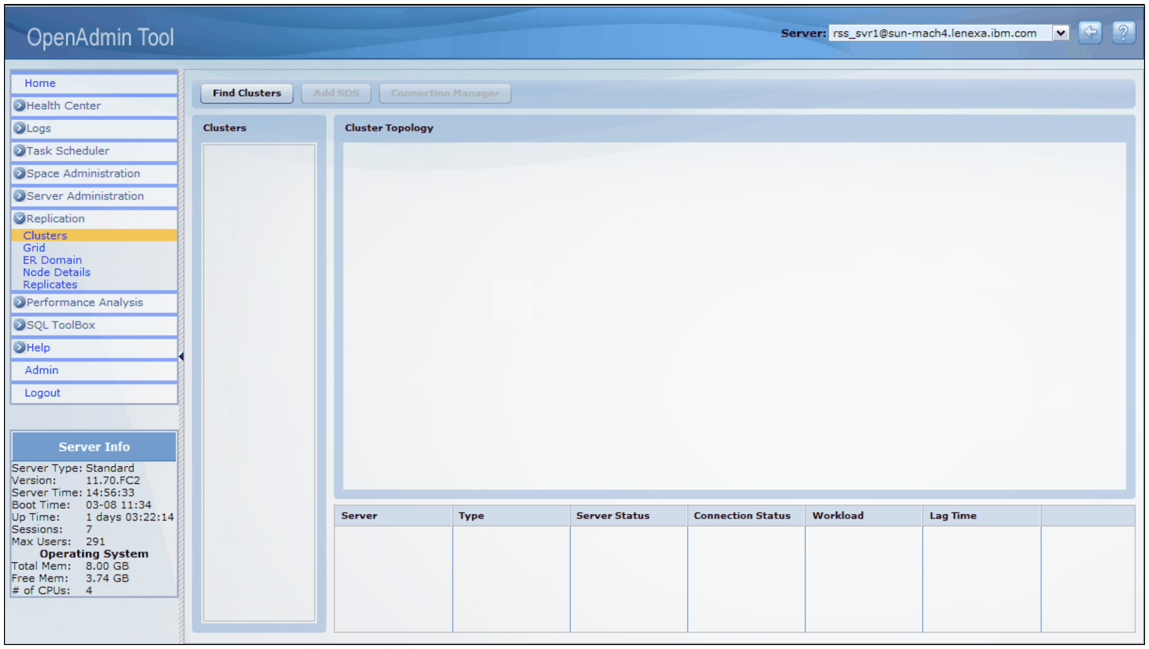
Informix cluster detection and layout retrieval

The first step of monitoring the Informix clusters is to initiate a cluster layout retrieval. You can start the retrieval after successfully logging in to a database server that has already been defined in the OAT. This server should be attached to a cluster environment.

Perform the following steps to register a new Informix cluster environment in the OAT registry:

1. Log in to an already defined database server in the OAT.

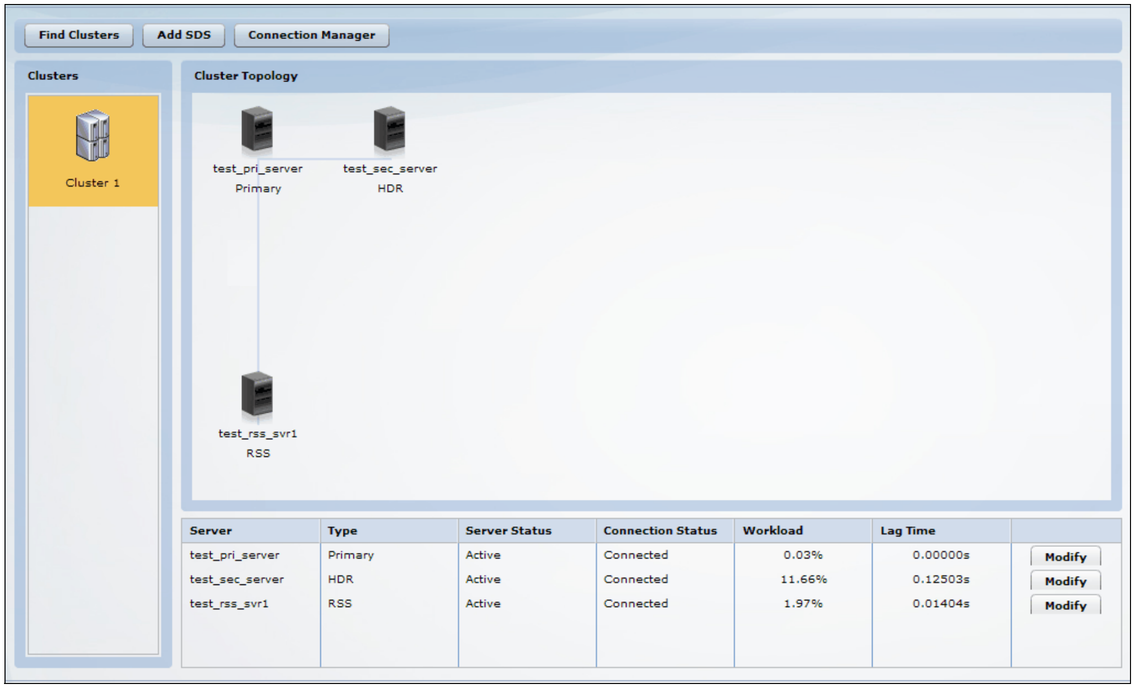
2. Select **Replication** → **Clusters**. Initially, there are no clusters defined and listed, as shown in Figure 7-32. Click **Find Clusters**.



```
Server Type: Standard
Version:     11.70.FC2
Server Time: 14:56:33
Boot Time:   03-08 11:34
Up Time:     1 days 03:22:14
Sessions:    7
Max Users:   291
   Operating System
Total Mem:   8.00 GB
Free Mem:    3.74 GB
# of CPUs:   4
```

Figure 7-32 Empty cluster topology

3. OAT shows the currently defined database server in the cluster. Figure 7-33 gives an example. You can see the cluster layout and also a table of the associated database servers, their server types, and the currently defined workload.



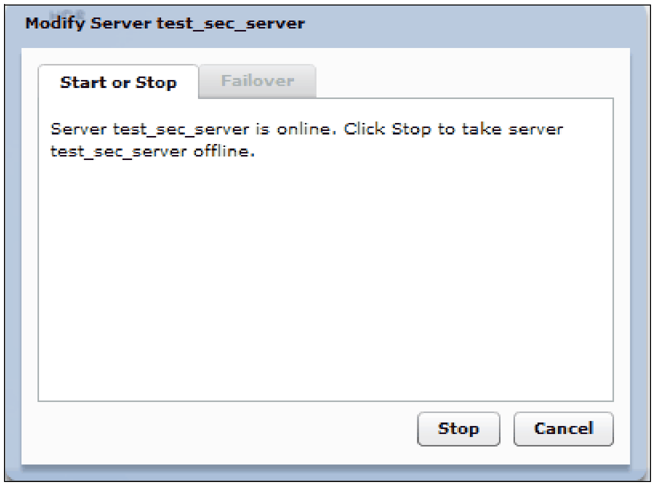
Server	Type	Server Status	Connection Status	Workload	Lag Time	
test_pri_server	Primary	Active	Connected	0.03%	0.00000s	Modify
test_sec_server	HDR	Active	Connected	11.66%	0.12503s	Modify
test_rss_svr1	RSS	Active	Connected	1.97%	0.01404s	Modify

Figure 7-33 OAT MACH11 cluster layout

Changing the database server mode

There are situations when you want to take server offline, for example, scheduled downtimes to perform maintenance on a database server host machine.

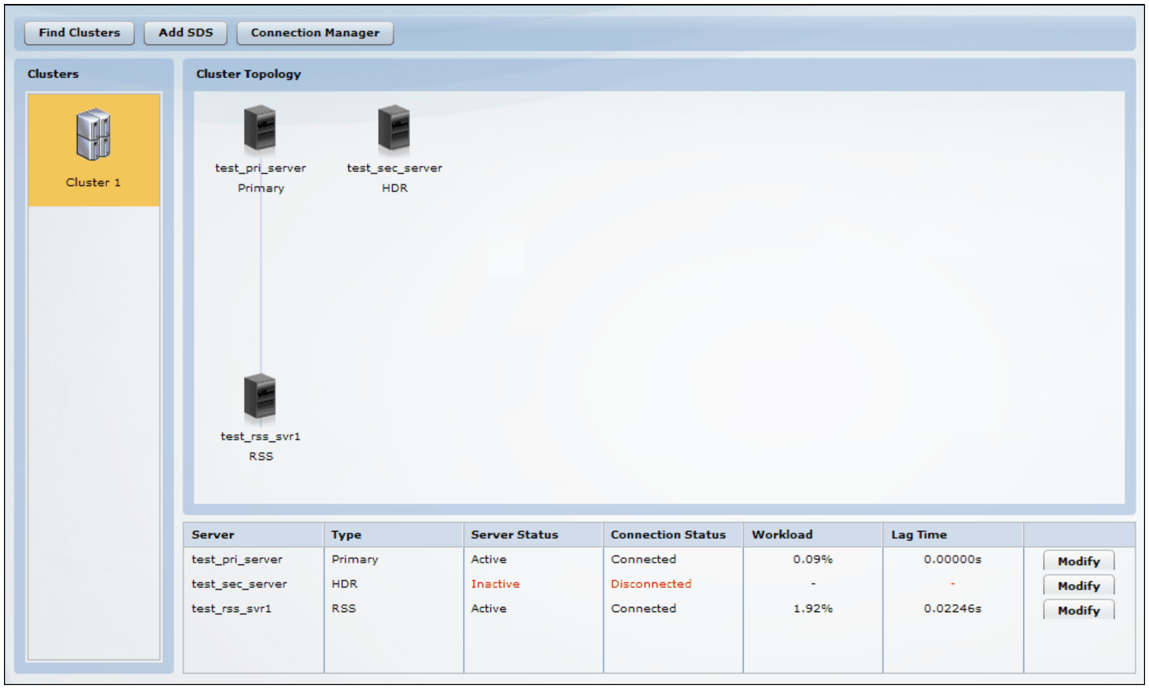
You can use the OAT interface to change the database server mode. To complete this task, click **Modify** on the server entry. Another window opens and shows the current status and an acknowledgement for the requested operation, as shown in Figure 7-34.



Server test_sec_server is online. Click Stop to take server test_sec_server offline.

Figure 7-34 Modifying the database server mode

After the request is executed, refresh the status information of the current cluster by clicking the cluster icon again. Figure 7-35 shows the new status of the server.



Server	Type	Server Status	Connection Status	Workload	Lag Time	
test_pri_server	Primary	Active	Connected	0.09%	0.00000s	Modify
test_sec_server	HDR	Inactive	Disconnected	-	-	Modify
test_rss_svr1	RSS	Active	Connected	1.92%	0.02246s	Modify

Figure 7-35 Server status after changing the mode

Adding a new SDS to an existing cluster environment

Besides high availability, the other main usage of cluster is the distribution of clients across available database server resources. There are situations where an new node can be added to an existing MACH11 cluster to share the workload. Adding an additional SDS server is easy and quick, as you only have to clone the configuration file and create temporary dbspaces for expected temporary table activity, and then you are ready to start the server. We discuss adding an SDS node to an existing cluster environment in Chapter 10, “Shared Disk Secondary servers” on page 395.

SLA and failover strategy maintenance for Connection Manager

In addition to managing and monitoring the particular server in an Informix cluster environment, OAT also provides the functionality for maintaining the SLA and FOC of the existing Connection Managers. For more details, refer to Chapter 11, “Connection management in an Informix environment” on page 411.

7.6.2 The onstat command

The onstat command is the command-line tool for finding the status of a cluster. The **onsat -g cluster** command combines the functionality of various **onstat** output, such as:

- **onstat -g dri** for High Availability Data Replication (HDR)
- **onstat -g sds** for Shared Disk Secondary (SDS)
- **onstat -g rss** for Remote Standy Secondary (RSS)

The output of the **onstat -g cluster** command differs slightly depending on whether the command is run on the primary server or on one of the secondary servers.

Example 7-3 shows the output of **onstat -g cluster** command when run on the primary and one of the secondary servers. For detailed information, you can also run **onstat -g cluster verbose**.

Example 7-3 onstat -g cluster

```
#On Primary Server
hp-mach3:(TEST):/opt/ibm>onstat -g cluster

IBM Informix Dynamic Server Version 11.70.FC2 -- On-Line (Prim) -- Up 00:43:24 -- 1096648 Kbytes

Primary Server:test_pri_server
Current Log Page:6,521
Index page logging status: Enabled
Index page logging was enabled at: 2011/03/09 22:28:33


Server          ACKed Log    Supports     Status
                (log, page)  Updates
test_sds_svr1   6,521        Yes          SYNC(SDS),Connected,Active
test_sec_server 6,521        Yes          SYNC(HDR),Connected,On
test_rss_svr1   6,521        Yes          ASYNC(RSS),Connected,Active

#On RSS Server
hp-mach4:(TEST):/opt/ibm>onstat -g cluster

IBM Informix Dynamic Server Version 11.70.FC2 -- Updatable (RSS) -- Up 17:12:40 -- 1113032 Kbytes

Primary Server:test_pri_server
Index page logging status: Enabled
Index page logging was enabled at: 2011/03/09 22:28:33


Server          ACKed Log    Supports     Status
                (log, page)  Updates
test_rss_svr1   6,527        Yes          ASYNC(RSS),Connected,Active
```

8

High Availability Data Replication

There are many ways to achieve high availability today. However, it is often difficult to actually define the term *high availability* because it is a subjective term. For example, sometimes high availability is not associated with disaster recovery, but other times the two are seen as the same thing. But one can clearly differentiate between the two depending on failover time. In the case of high availability, the failover should be quick. Typically, these failures are caused by hardware, software, infrastructure, and human errors. Disaster recovery, on the other hand, is where an external factor causes a failure and the entire data center needs to recover from it.

In simple terms, a system is highly available if it can avoid a single point of failure. Single point of failures are undesirable in any system whose goal is high availability, be it a network, software application, or the system itself. Systems are made highly available by adding redundancy to all potential points of failure. Redundancy can be achieved at the component level, at the system level (one or more machines), or at the site level (data center).

In this chapter, we discuss the redundancy that High Availability Data Replication (HDR) provides at a database server level. This discussion will give you a better understanding of what HDR is and how it can be used.

8.1 What is HDR and how it works

It might be best to first describe what HDR is *not*. For example, HDR is not a variation of disk mirroring. In a mirrored environment, one system controls two sets of storage containers and manages the I/O operations of both sets. Although HDR requires two sets of storage, each one is managed independently by a separate physical server. When data definition language (DDL) and logged data manipulation language (DML) statements are executed on the primary server, those operations are replicated on the secondary server at a predefined interval. Because most companies use HDR for fault tolerance, the interval is usually set to *immediate*. But it can be set to any reasonable positive number of seconds.

As early as Informix Version 6, Informix adopted HDR technology, which is fully integrated within the data server. HDR is easy to set up and administer and does not require any additional hardware or software for automatically handling server or disk failures. Client application performance is improved by means of localized database server connections.

HDR maintains two identical Informix database servers with similar configurations and operating systems, as shown in Figure 8-1. HDR employs a log record shipping technique to transfer the logical log records from the primary server to the secondary server. The secondary server is in perpetual roll-forward mode so that data on the secondary server remains current with data on the primary server. The secondary server supports read/write access to data, allowing database administrators to spread workload among multiple servers.

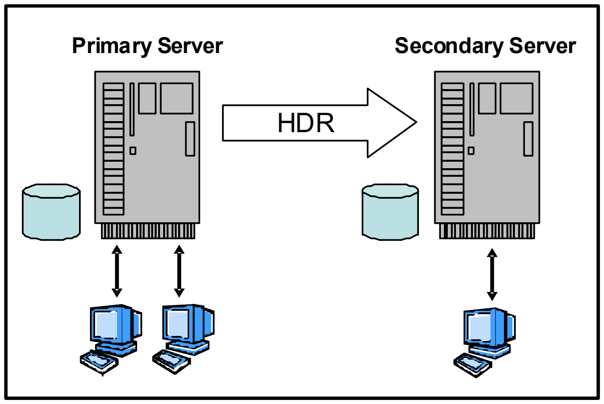


Figure 8-1 High Availability Data Replication

The secondary server can be configured to operate in SYNC (synchronous) or ASYNC (asynchronous) mode. In SYNC mode, HDR guarantees that when a transaction is committed on the primary server, its logs have been transmitted to the HDR secondary server. In ASYNC mode, transaction commitment on the primary and transmission of updates to the secondary are independent. This can provide better performance, but brings with it the risk of possibly losing transactions.

Stand-alone HDR provides automatic failover using the DRAUTO configuration parameter. However, as a best practice, use Connection Manager for automatic failover and client redirection, as it is more reliable than DRAUTO. For more details, refer to Chapter 11, "Connection management in an Informix environment" on page 411.

With DRAUTO set, if the primary server fails, the HDR secondary server automatically takes over and switches to a standard or primary server. When the original primary server becomes available, it is synchronized when replication is re-established.

Stand-alone HDR can support the basic client redirection feature, which makes failover transparent to the application. To use basic client redirection, the primary and secondary servers must be defined as a server group in the `SQLHOSTS` file, as shown in Example 8-1.

Example 8-1 Sample sqlhosts file

```
cluster group - - i=1
pri_server onsoctcp pri_host 14001 g=cluster
sec_server onsoctcp sec_host 14002 g=cluster
```

Applications use the server group name *cluster* to connect to the database server. The network layer and the client server protocol ensures that the client is always connected to the primary server in the group. If the primary server fails and the secondary server becomes the new primary server, clients connecting to the group are automatically connected to the new primary server. An application cannot be redirected to secondary server using this basic client redirection method.

8.2 How to set up HDR

In setting up HDR, an almost-true mirror of the primary system is created. Why is it not a full mirror? It is not because data stored in non-logged databases, tables, and BLOB spaces is not replicated. Updates to such data is written directly to storage without passing through the logical logs and, as a result, is not captured for transfer to the secondary. With this one caveat, in this chapter we refer to an HDR pair as an exact mirror or full replica.

8.2.1 Prerequisites for HDR to work

Because HDR operates as a mirror, it has a series of prerequisites, some stringently enforced, which must be met. A second set of storage devices has already been mentioned. These devices, managed by the secondary server, must be identical to those configured within the primary instance in terms of number, size, and path. If the primary uses a RAID LUN, for example, it is not required that the secondary also use RAID; it just needs to present to the instance the same number and size of storage devices. The devices must be created and available on the same PATH on the secondary as on the primary. This setup is most easily accomplished using symbolic links, such as the PATH parameters in the *onspaces* command while creating chunks and dbspaces.

Two other stringently enforced requirements are that both servers must be using the same OS and Informix binaries. While there can be a little variation at the OS level (for example, having different release levels of the same OS version, such as IBM AIX 5L™ V5.2 and AIX 5L V5.3), it is not possible to set up HDR between two different operating systems or between two different versions of the same operating system. The Informix version match check is rigidly enforced. The version must be identical all the way to the sub-version identifier, such as 11.70.UC1.

It is helpful, though not necessarily required, that the computing hardware be identical as well. This is particularly important if the intention for the secondary is to provide full user support while the primary is unavailable. While HDR will fail over to the secondary instance, even if it only has 25% of the configured shared memory and CPU VPs as the primary, in the event of an HDR failure condition and a switch to the secondary occurs, users will notice response and performance differences due to the reduced configuration of the secondary. If the intention for the secondary is just to bridge the gap while the primary is unavailable and only the most critical user processes will be redirected to the secondary, there might not be a noticeable difference.

Another argument for server hardware parity is that the secondary server can be used for read-only operations while in a functional HDR pair. As such, read-only applications such as reports and so on, can be executed against the secondary instance, reducing the load on the primary server/instance. The benefit is that both read/write and report applications should see a decrease in execution time with the processing load distributed across two instances instead of just one. If the secondary only has a minimal hardware configuration, it could take just as long, if not longer, for reporting/read-only applications to execute than if they were executed against the more fully configured primary instance.

8.2.2 Prerequisites to set up HDR

From the database server perspective, the two instances involved in replication should be configured as similarly as possible, particularly if the secondary will be expected to fully replace a down primary instance. The best way to configure the secondary instance is to copy the `$ONCONFIG` file from the primary to the secondary and change the parameter specific to secondary server, such as DBSERVERNAME and DBSERVERALIASES and MSGPATH. This action ensures that, among other things, the replication configuration parameters, such as DRAUTO and DRINTERVAL, are set identically so each instance takes the appropriate action in the event of a failure.

The following **`onconfig`** parameters must be set identically on both the primary and the secondary server:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT

The `INFORMIXSQLHOSTS` file should contain connection information of both servers in the HDR pair.

HDR will not work with shared memory connections. A DBSERVERNAME or DBSERVERALIAS must be running TCP/IP connections to accommodate HDR.

Both machines on an HDR pair must be trusted. Modify the `/etc/hosts.equiv` or the `.rhosts` file for the informix user ID to enable trusted communication. Both instances must be on the same Wide Area Network (WAN) or Local Area Network (LAN).

If you are using onbar as your archiving mechanism, then the storage manager version must be identical on both the primary and the secondary. Also, ensure that the XBSA libraries installed on both servers are compatible.

When all the prerequisites have been met, the HDR initialization process can begin.

8.2.3 Mandatory onconfig parameters

There are several `onconfig` parameters that define the HDR behavior and its performance. We discuss and describe those parameters in this section.

DRAUTO

The intent of this parameter is a noble one in that it determines what action the secondary instance should take in the event of a replication failure. Depending on how this parameter is set, in a failure the secondary can perform one of the following actions:

- 0 (OFF): Remain in read-only mode until manually switched to standard or primary mode.
- 1 (RETAIN_TYPE): Automatically switch to primary mode, process transactions from users, and then revert back to secondary mode when the original primary returns. As part of the change back to secondary, it transfers its transaction records to the primary so they are logically consistent.
- 2 (REVERSE_TYPE): Automatically switch to primary mode and process transactions. When the original primary returns, the original primary converts to secondary mode and receives transaction information from the new primary so both are logically consistent.
- 3: Specifies that Connection Manager will perform the automatic failover (switch secondary to primary mode).

An issue with this built-in arbitrator (DRAUTO set to 1 or 2) is that it cannot differentiate between an actual hardware failure versus a network failure. Although hardware failure is certainly possible, it is far more common for a network to fail and cause replication failure. If a secondary switches to primary mode as a result of network outage between a primary and secondary server, that action can cause serious data integrity problems due to there being two primaries in an HDR pair.

For this reason, as best practice, set DRAUTO to 3 and use Connection Manager for automatic failover. When Connection Manager is configured as a failover arbitrator, it monitors the heartbeat of the cluster (HDR pair, in this case) and initiates failover only when it has made sure that primary is really down or not reachable by any other cluster nodes. If a network outage occurs between the primary and Connection Manager, then Connection Manager will ask the secondary server to confirm whether or not it lost the connection with the primary. If that secondary server confirms that it also lost the network along with the primary, then only Connection Manager will start the failover process. This type of dual confirmation enables Connection Manager to distinguish between hardware failure versus network failure.

DRINTERVAL

DRINTERVAL specifies the maximum number of seconds between flushes of the HDR data buffers.

HDR has two main modes of operation:

- Synchronous
- Asynchronous

To understand the difference, it us helpful to be aware of how updates propagate from the primary to the secondary. Updates made to the primary server are replicated on the secondary server by having the primary server send all its logical-log records to the secondary server as the logs are generated.

HDR synchronous updating

When DRINTERVAL is set to -1, data replication to the HDR secondary server occurs synchronously. As soon as the primary database server writes the logical-log buffer contents to the HDR buffer, it sends those records from the buffer to the HDR secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the HDR secondary database server that the records were received.

HDR asynchronous updating

If you set DRINTERVAL to any value other than -1, data replication occurs asynchronously to the HDR secondary server. The primary database server flushes the logical-log buffer after it copies the logical-log buffer contents to the HDR buffer. Independent of that action, the primary database server sends the contents of the HDR buffer across the network when one of the following conditions occurs:

- The HDR buffer becomes full.
- The time interval specified by DRINTERVAL on the HDR primary has elapsed since the last time that the HDR replication buffers have been flushed.

DRTIMEOUT

The DRTIMEOUT configuration parameter specifies the interval for which either database server waits for a transfer acknowledgment from the other. If the primary database server does not receive the expected acknowledgment, it adds the transaction information to the file named in the DRLOSTFOUND configuration parameter. If the secondary database server receives no acknowledgment, it changes the data-replication mode as the DRAUTO configuration parameter specifies.

A database server uses following formula to detect a timeout:

DRTIMEOUT = wait_time / 4

In this formula, wait_time is the length of time, in seconds, that a database server in a High Availability Data Replication pair must wait before it assumes that a High Availability Data Replication failure occurred.

For example, suppose you determine that wait_time for your system is 160 seconds. Use the preceding formula to set DRTIMEOUT as follows:

DRTIMEOUT= 160 seconds/ 4 = 40 seconds

Detection of HDR failure

The database server interprets either of the following conditions as an HDR failure:

- The specified timeout value was exceeded. During normal HDR operation, a database server expects confirmation of communication from the other database server in the pair. Each database server in the pair has an **`onconfig`** parameter, DRTIMEOUT, that specifies a number of seconds. If confirmation from the other database server in a pair does not return within the number of seconds that DRTIMEOUT specifies, the database server assumes that an HDR failure has occurred.
- The other database server in the primary-secondary pair does not respond to the periodic messaging (pinging) attempts over the network. The database servers ping each other regardless of whether the primary database server sends any records to the secondary database server. If one database server of a primary-secondary pair does not respond to four sequential ping attempts, the other database server assumes that an HDR failure has occurred.

  Each database server in the pair sends a ping to the other database server in the pair when the number of seconds specified by the DRTIMEOUT parameter on that database server has passed.

Checkpoints

A checkpoint on the primary database server completes only after it completes on the secondary database server. If the checkpoint does not complete within the time that the DRTIMEOUT configuration parameter specifies, the primary database server assumes that a failure has occurred.

DRLOSTFOUND

DRLOSTFOUND specifies the path name to the `dr.lostfound.timestamp` file. This file contains transactions committed on the primary database server but not committed on the secondary database server when the primary database server experiences a failure and the secondary takes over as the new primary. This situation could result in a lost transaction. This parameter is not applicable if updating between the primary and secondary database servers occurs synchronously (that is, if DRINTERVAL is set to -1).

To reduce the risk of a lost transaction without running data replication in synchronous mode, use unbuffered logging for all the databases. This method reduces the amount of time between the writing and transfer of the transaction records from the primary database server to the secondary database server.

8.2.4 Optional onconfig parameters

In this section, we discuss the **`onconfig`** parameters that also define the HDR behavior, but can be set optionally.

DRIDXAUTO

Specifies whether the primary server automatically starts index replication if the HDR secondary server detects a corrupted index. To enable automatic index replication, set the value of the DRIDXAUTO configuration parameter to 1.

Note: You can update the value of DRIDXAUTO dynamically by using the **`onmode -d idxauto`** command. However, this command will not change the value of the DRIDXAUTO parameter in the `onconfig` file.

If an index on an HDR secondary database server becomes corrupt and needs to be rebuilt, either of the following actions can be taken:

- Manually replicate the index from the primary server to the secondary server.
- Let the secondary server automatically replicate the index if DRIDXAUTO is enabled.

If this functionality is enabled, when one of the threads on the secondary database server detects a corrupt index, the index is automatically replicated to the secondary database server. Restarting index replication can take up to the amount of time specified in seconds in the DRTIMEOUT configuration parameter.

Sometimes it may be desirable to replicate an index manually, for example, when index repair must be postponed because the table is locked. To be able to manually replicate an index on the HDR secondary server, turn off the automatic replication feature.

To turn off the automatic index replication feature, either:

- Set **`onmode -d idxauto`** to off.
- Set the DRIDXAUTO configuration parameter to 0.

If **`onmode -d idxauto`** is set to off or DRIDXAUTO is set to 0 and the secondary server detects a corrupt index, you can manually replicate an index on the HDR secondary server by issuing an **`onmode -d index`** command in the following format:

```
onmode -d index database:[ownername].table#index
```

In the case of a fragmented index with one corrupt fragment, the **`onmode -d idxauto`** option only transfers the single affected fragment, whereas the **`onmode -d index`** option transfers the whole index.

The `online.log` file produced by the secondary server contains information about any replicated index.

LOG_INDEX_BUILDS

With Informix 11, there is a new algorithm (Index Page Logging) to handle the transfer of the index from the primary to the secondary server. When Index Page Logging is activated, the primary sends the index to the HDR secondary through the logs. The algorithm transfers the index pages to the secondary while creating the index.

Index Page Logging can be turned on by setting the LOG_INDEX_BUILDS parameter to 1. It can be enabled dynamically by running the following command:

```
onmode -wf LOG_INDEX_BUILDS=1
```

LOG_STAGING_DIR

Checkpoints between a primary server and an HDR secondary server are synchronous, regardless of the value of DRINTERVAL. A checkpoint on the primary database server completes only after it completes on the secondary database server. If checkpoint processing on the HDR secondary server takes long time, for whatever reason, it can affect the transaction processing on the primary server, as primary has to wait for the HDR secondary to start accepting new logs, effectively disabling the non-blocking checkpoint functionality of Informix on the primary server.

With Informix Version 11.50.xC6, you can configure an HDR secondary server to allow true non-blocking checkpoint on the primary. When configured, the primary server does not have to wait for the checkpoint to complete on the HDR secondary server.

When the HDR secondary server encounters a checkpoint, it enters a buffering mode. While in buffering mode, the secondary server stages log data received from the primary server into files in the staging directory set by the LOG_STAGING_DIR parameter. This action enables the HDR secondary to accept new logs from the primary while processing a checkpoint. This action also enables the primary to continue transaction processing by sending logs to the HDR secondary.

When the HDR secondary server completes checkpoint processing, the server enters a drain mode. In this mode, the HDR secondary server reads data from the staged log files and also receives new data from the primary server. After the all staged log files are applied, the HDR secondary server resumes normal operation.

You enable non-blocking checkpoints in HDR pair by setting the LOG_STAGING_DIR configuration parameter on the HDR secondary server, and LOG_INDEX_BUILDS on both the primary server and the HDR secondary server. The value for LOG_INDEX_BUILDS should be the same on both database servers. You can dynamically change the value of LOG_STAGING_DIR parameter by running the following command:

```
onmode -wf LOG_STAGING_DIR=/path/to/staging/dir
```

Where log records are stored on the HDR server

The HDR secondary server creates additional directories named `ifmxhdrstage_##` in the directory specified by LOG_STAGING_DIR, where ## is the instance specified by SERVERNUM. The directories are used to store the logical logs sent from the primary server during checkpoint processing. The files within `ifmxhdrstage_##` are purged when no longer needed.

Non-blocking checkpoints and updates on the secondary

You should consider the interaction between secondary server updates and non-blocking checkpoints on the HDR secondary servers. If the HDR secondary server receives an update request, the updates are not applied until the HDR secondary server processes the corresponding log records. When non-blocking checkpoints are enabled on the HDR secondary server, a delay in the application of data on the secondary server might occur because log data is staged at the secondary server due to checkpoint processing.

To view information about non-blocking checkpoints on primary servers and on HDR secondary servers, run **`onstat -g dri ckpt`**.

ENCRYPT_HDR

ENCRYPT_HDR enables or disables HDR encryption. Enabling HDR encryption provides a secure method for transferring log data from one server to another in an HDR pair. HDR encryption works in conjunction with Enterprise Replication (ER) encryption. However, it is not necessary to have ER encryption enabled for HDR encryption. HDR encryption works whether or not ER encryption is enabled.

HDR and ER share the same encryption configuration parameters, that is, ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE, and ENCRYPT_SWITCH.

OFF_RECVRY_THREADS

OFF_RECVRY_THREADS are the number of recovery threads that apply the log records on the secondary server. If care is not taken, then there will be an imbalance of the usage of the recovery threads, which will result in a bottleneck. Care must be taken to reduce an imbalance in recovery thread usage or you will end up underutilizing the resources on the secondary. This situation can result in a backflow that will impact the primary server's performance.

To avoid recovery thread imbalances, complete the following actions:

- Run **`onstat -g cpu`** and look at the column called CPU Time for recovery threads (xchg_##). If one of the recovery threads is doing the bulk of the work, then you have a potential imbalance. Make OFF_RECVRY_THREADS at least three times the number of CPU VPs (minimum of 11). These are the threads that apply the log records on the secondary, so do not underprovision or replication performance will suffer. Consider using a near prime number of recovery threads (that is, a number not divisible by 2, 3, 5, 7, or 11).
- Use round-robin fragmentation on most active tables and indexes. By using fragmentation, the recovery threads are better utilized. Fragmentation by expression will generally not be as useful.
- Do not put an index in the same partition as the table. Fragment indexes are frequently updated, so doing putting the index in the same partition as the table will increase the probability of balancing the log apply workload.

UPDATABLE_SECONDARY

By default, a secondary server is initialized in Read-Only mode, which means database updates are not allowed on the secondary server. However, you can use the UPDATABLE_SECONDARY parameter to enable database updates on the secondary server. The value of the parameter specifies the number of connections to establish between the primary and secondary servers to process database updates. After a secondary server is initialized as an updatable secondary server, it can support the COMMITTED READ, and COMMITTED READ LAST COMMITTED transaction isolation levels.

8.2.5 Steps to set up HDR

In this section, assume that you have completed the prerequisites discussed in 8.2.1, "Prerequisites for HDR to work" on page 364. The basic idea behind the HDR setup is that you take a level 0 (zero) archive of the primary instance and, using that archive, perform a physical restore on secondary instance. After the restore is complete, you define the server type and you are done.

The backup and restore steps given below are for the ontape utility, but you can use the onbar utility or external backup and restore as well.

Steps on the primary instance

Complete these steps on the primary server:

1. Take a level 0 archive using the following command:

   ```
   ontape -s -L 0
   ```

2. Define an instance as primary server using the following command:

   ```
   onmode -d primary secondary_server
   ```

Steps on the secondary instance

On the secondary server, complete these steps:

1. Make sure that all chunk paths are created and have the correct permissions.
2. Complete a physical restore of the primary instance level-0 archive using the following command:

   ```
   ontape -p
   ```

3. Define an instance as a secondary instance using the following command:

   ```
   onmode -d secondary primary_server
   ```

8.3 Recovering from an HDR failure

As mentioned earlier in “DRAUTO” on page 366, the DRAUTO parameter plays a critical role in automatic failover in an HDR environment. As a best practice, use Connection Manager as a failover arbitrator to handle automatic failover and application redirection in an HDR environment.

The scenario discussed in this section is based on the assumption that DRAUTO is set to 3. For steps about how to recover from an HDR failure using other DRAUTO settings, go to the following address:

http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.admin.doc/ids_admin_0956.htm

Whether you have a planned or unplanned outage, that is, when a primary fails, the failover arbitrator (Connection Manager in this case) switches the HDR secondary to primary mode. When you are ready to restart the offline server, you should start it as a secondary server. To restart the offline server as secondary server, complete the following steps:

- Start the server by completing a physical recovery by using **`oninit -PHY`**.
- Define the server type as secondary by using **`onmode -d secondary primary_server`**.

After the secondary reconnects to the primary, the primary sends the necessary log records to the secondary to synchronize the data.

If the primary does not have the log records (because the logical logs were reused) that the HDR secondary needs to get back to a synchronized state, you will see a message, as shown in Example 8-2, in the secondary server log. You have to then roll forward the logical logs from the log backup of the primary server. For example, you can run **`ontape -l`** to roll forward the logical logs.

Example 8-2 Replay the logical logs

```
17:45:57  DR: Start failure recovery from tape ...
```

After the logs are rolled forward from the backup, the primary and secondary will be synchronized.

After the pair is operational, and you want to switch the roles of the database server, shut down the current primary and Connection Manager switches the HDR secondary to primary mode automatically. Repeat the above process to start an offline instance as a secondary server.

8.4 Monitoring and troubleshooting

Various user interfaces are available to monitor, administer, and troubleshoot an HDR environment, such as:

- OpenAdmin Tool (OAT) for web-browser based administration
- Onstat and onmode for command-line administration
- System catalog (sysdri table in sysmaster database) to automate monitoring through SQL interface
- Event alarms to automate administration via alarm handler

8.4.1 OpenAdmin Tool

OpenAdmin Tool (OAT) provides a web-browser based interface to DBAs to perform remote administration and monitoring of Informix cluster components. Figure 8-2 shows how OAT provides a complete view of the cluster topology and Figure 8-3 shows the current state of the cluster. In this example, the cluster includes a primary and an HDR secondary only.

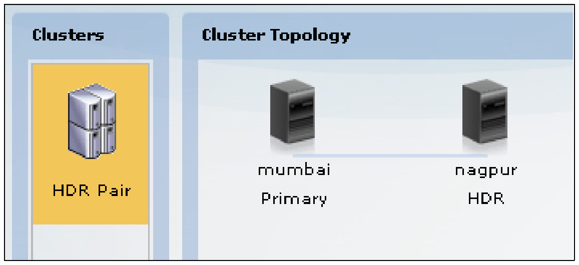
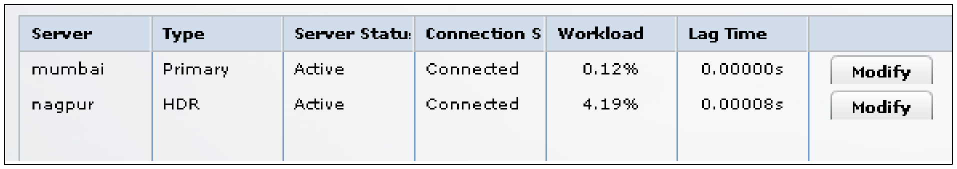


Figure 8-2 Cluster topology



Server	Type	Server Statu:	Connection S	Workload	Lag Time	
mumbai	Primary	Active	Connected	0.12%	0.00000s	Modify
nagpur	HDR	Active	Connected	4.19%	0.00008s	Modify

Figure 8-3 Cluster state

8.4.2 onstat and onmode

There are two **onstat** options that can be used to look at the current state of an HDR pair:

- **onstat -g dri** (for HDR specific statistics)
- **onstat -g cluster [verbose]** (for cluster level statistics)

Example 8-3 shows the output of the **onstat -g cluster** command on the HDR secondary server.

Example 8-3 onstat output

```
$ onstat -g cluster

IBM Informix Dynamic Server Version 11.70.UC1     -- Updatable (Sec) -- Up 00:08:26 -- 152348 Kbytes

Primary Server:mumbai
Index page logging status: Enabled
Index page logging was enabled at: 2011/02/22 20:19:37


Server ACKed Log    Supports     Status
       (log, page)  Updates
nagpur 4,58         Yes          ASYNC(HDR),Connected,On


$ onstat -g cluster verbose

IBM Informix Dynamic Server Version 11.70.UC1     -- Updatable (Sec) -- Up 00:08:31 -- 152348 Kbytes

Data Replication at 0x4b42e1a0:
  Type           State        Paired server        Last DR CKPT (id/pg)    Supports Proxy Writes
  HDR Secondary  on           mumbai                        4 / 57         Y

  DRINTERVAL   30
  DRTIMEOUT    30
  DRAUTO       0
  DRLOSTFOUND  /work/nilesho/testnode/sqldist/etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0
  Backlog      0
```

You can use the onmode utility to change the behavior of the server and switch modes and dynamically modify certain parameters. Table 8-1 shows some of this functionality.

Table 8-1 HDR uses of onmode

onmode option and arguments	Explanation
`onmode -d idxauto`	This command enables automatic index replication when an index on a secondary server becomes corrupt.
`onmode -d index [options]`	This command replicates an index (if it becomes corrupt when crossing from the primary to the secondary).
`onmode -d primary` ***secservername***	This command turns the server into an HDR primary, and tells it to look for the secservername as the secondary server.
`onmode -d secondary` ***primservername***	This command turns the server into an HDR secondary, and tells it to look for the primservername as the secondary server.
`onmode -d standard`	This command turns HDR off (on the primary) and turn the engine into a stand-alone server.

8.4.3 Online log

The online log contains important and informational messages pertaining to HDR. Table 8-2 lists some of the messages that can be found in the online log to indicate the status of HDR.

Table 8-2 Common online log messages and meanings

Online log message	Description
`DR: DRAUTO is 0 (Off).`	DRAUTO's current setting is displayed whenever shared memory is initialized (this is done even in STANDARD mode).
`DR: ENCRYPT_HDR is 0 (HDR encryption Disabled).`	The current setting of ENCRYPT_HDR is shown. For more information about encryption, refer to "ENCRYPT_HDR" on page 372.
`DR: Reservation of the last logical log for log backup turned on.`	When o**`nmode -d primary sec_servername`** is run, Informix will now save the last free logical log so that a logical log backup can be run if necessary.

Online log message	Description
`DR: new type = primary, secondary server name = secondary.`	This message is inserted into the online log whenever HDR is turned on.
`DR: Unable to change server type.`	Whenever an illegal HDR mode change is tried, this message will be placed in the online log.
`DR: DRAUTO invalid, now set to 0.`	If DRAUTO is set to an invalid value, Informix will automatically change it to 0. This message will also be placed in the online log.
`DR: Force timeout.`	If the two servers are unable to communicate for the duration specified by DRTIMEOUT, then this message will be placed in the online log.

There are many other messages that could be found in the log. The DBA should monitor the online log to be aware of any problems.

8.4.4 The sysmaster database

The sysmaster database table, sysdri, is basically a reflection of the information that can be found with the **`onstat -g dri`** command. Table 8-3 shows the associated columns that correspond closely to the fields from an **`onstat -g dri`** command.

Table 8-3 sysdri database

Column	Description
type	As mentioned, there are three types possible with HDR: ► Primary ► Secondary ► Standard
state	Tells whether the replication is on or off.
name	The DBSERVERNAME or DBSERVERALIAS that this instance is connecting to for replication. If the sysdri table is reflected on the primary, the output will correspond to the first column in sqlhosts that points to the secondary instance.
intvl	DRINTERVAL setting.
timeout	DRTIMEOUT setting.
lostfound	DRLOSTFOUND setting.

Column	Description
drauto	DRAUTO setting.
dridxauto	DRIDXAUTO.

Notice that the checkpoint information is *not* included with this output.

8.4.5 Event alarms

Informix provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server environment. This mechanism is the event alarm feature. Events can be informative (for example, `Backup Complete`) or can indicate an error condition that requires your attention (for example, `DR: Turned off on primary server`).

Using an ALARMPROGRAM **`onconfig`** parameter, you can capture events. To capture events, set ALARMPROGRAM to `$INFORMIXDIR/etc/alarmprogram.sh` or `%INFORMIXDIR%\etc\alarmprogram.bat` (on Windows) and modify the file to handle HDR event alarms of class ID 15.

For more details about event alarms, go to the following address:

`http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.adref.doc/ids_adr_0668.htm`

9

Remote Secondary Standby database servers

Remote Secondary Standby (RSS) servers advance the functionality of the High Availability Data Replication (HDR) servers we discussed in Chapter 8, “High Availability Data Replication” on page 361. These servers provide a one to many relationship between a primary server and secondary servers. The features of RSS that make it an attractive solution for high availability include a simple setup and flexibility in attaching and detaching new servers into the existing cluster. These features also allow the existing cluster to add RSS servers as additional temporary or long term resources for better workload balancing, that is, so that the client applications are distributed according to their resource and execution time requirements.

In this chapter, we discuss the technical details about setting up a new or updating an existing RSS based MACH11 cluster. We show various cluster monitoring facilities.

9.1 Why use RSS as a database cluster solution

Data availability and workload balancing are two major goals of cluster solutions. These features provide continuous data access with an optimal usage of all assigned database and host system resources. The Informix RSS supports both goals.

Database severs are frequently categorized by the data they contain and their relationships with other database servers. It takes more effort to build a database server that is defined as the master database server in a cluster. The master server is the data “headquarters” in the network and the relationship with the other database servers has to be planned and implemented.

The clustering should be machine and location independent. Machine independent means that the clustered systems are in the same location and can be in the same room. If a hardware failure occurs, the database server is switched to another host in the same location. Location independent means that the clustered systems are spread across remote sites. In case one location goes out of production due to a disaster, the other location can take over to provide the same data access consistency and functionality.

Figure 9-1 gives a conceptual view of the database cluster solution.

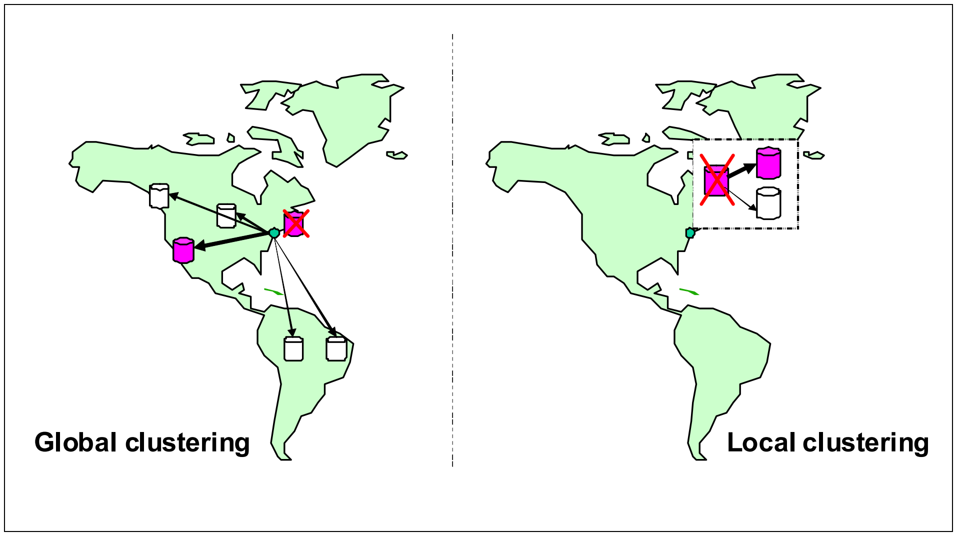


Figure 9-1 Clustering a database server with a data availability focus

A cluster provides an unique consistent view of the data across multiple server instances. With additional transaction management, such as through Connection Manager, to distribute applications, the resource utilization can be significantly improved. Using redundancy is a cost-effective method for load balancing capability, as shown in Figure 9-2.

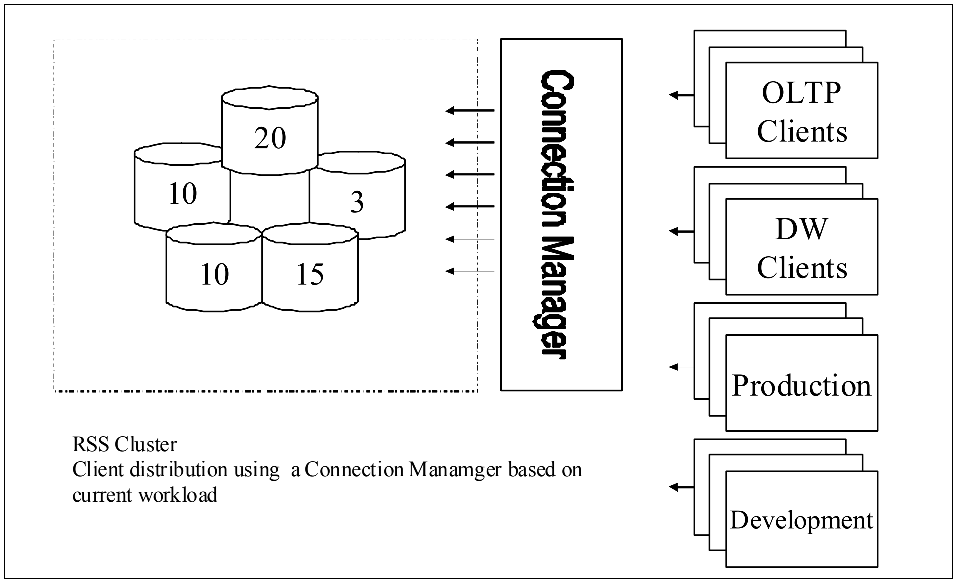


Figure 9-2 Workload balancing

Using an RSS server in a MACH11 cluster environment has two restrictions, that is, the primary and the secondary servers must have the same operating system and the same version of the database server.

If the requirement is to have a heterogeneous clustering systems containing different database server versions or to have the capability to execute operations such as DDL and DML across the cluster, consider Enterprise Replication and Flexible Grid instead.

9.2 How to establish an RSS database server

In the subsequent sections, we describe how to establish a new RSS database server. The discussion starts with an overview of the changes required on the database servers. We continue with a description of the steps for various setup approaches. We then discuss read-only and updatable secondary servers.

9.2.1 Preparing the database servers

From a database administration perspective, there are not many differences between adding a new database server into an existing MACH11 cluster and creating a new cluster out of a stand-alone database server. Before setting up an RSS server, perform the following tasks to prepare the database servers:

- Primary server:

  If this is first time setup of the RSS primary server, update the LOG_INDEX_BUILDS onconfig parameter. Set the value to 1. You can change the value dynamically without downtime by running the following command:

  ```
  onmode -wm LOG_INDEX_BUILDS=1
  ```

  This parameter defines that all index creation operations are tracked in the logical log and are applied on the RSS target to ensure database consistency. This configuration parameter should have been set if an RSS server already exists.

  Add the new RSS server to the `sqlhosts` file of the primary server to ensure successful communication between both servers.

- New RSS database server:

  Complete these tasks to prepare the new target server:

  - Install the same version of Informix on the primary server on the target server.
  - Copy the configuration files, such as `onconfig` and `sqlhosts`, from the primary server to the target server and apply the necessary changes for PATH or resource (amount of virtual processors and amount of assigned memory).
  - Make sure that the informix user is trusted on both servers. Change the `.rhosts` file or the `host.equiv` file as needed.
  - Set up all the chunk files to be similar to the files used by the primary server, that is, use the same file names and directory structure.

9.2.2 Setting up a new RSS database server

Setting up the new database server requires a copy of the current primary database server. The copy can be taken by using the ifxclone utility or a backup and restore solution provided by Informix. We briefly describe each approach.

Using the ifxclone utility

The easiest way to set up a new RSS secondary is by using the ifxclone utility. You must run this utility from the target system. The configuration files of the target server must be set up ahead of time or by using the **ifxclone** command options to copy the configuration files from the primary server. All the chunk paths used on the primary also must be on the target.

The items required by the **ifxclone** command are the host name, port number, and the host IP address of both the primary and the new RSS server. Here is an ifxclone example for establishing a new RSS:

```
ifxclone -S primary -I primary_host -P 1000 -t rss_secondary -i secondary_host
-p 1001 -T -L -d RSS
```

For more detailed information about the ifxclone utility, see Chapter 14, “Using ifxclone” on page 525.

Using the onbar backup and restore solution

Onbar is a physical database backup utility driven by the storage manager. You can take a database backup on the primary database server using onbar and restore the backup to the new RSS server. Afterwards, you establish communications between the new server pair by using the **onmode -d** command.

Complete the following steps:

- On the primary server:
  a. Run **onbar -b** to take a database backup.
  b. Run **onmode -d add RSS newrssserver** to register the new RSS server on the primary.
- On the new RSS server:
  a. Set up the complete environment for the database server, including the configuration files, such as `onconfig` and `sqlhosts`.
  b. Restore the backup taken on the primary database server by running **onbar -r -p**.
  c. Establish the connection between the new RSS server and the primary by running **onmode -d RSS primary_server**.
  d. Check the success of the operation by verifying the log file or running **onstat** to see if the new server is operational.

Using an external backup

An external backup allows the DBA to apply their own solution to back up the storage files maintained by the database server. This solution can be a copy or hardware mirroring.

Complete these steps to use external backups to set up an RSS server:

- On the primary database server:
  a. Block the database server by running **`onmode -c block`**.
  b. Back up externally all dbspaces of the primary server.
  c. Unblock the server by running **`onmode -c unblock`**.
  d. Register the new RSS server by running **`onmode -d add RSS secserver`**.
- On the new RSS server:
  a. Use the external restore mechanism to restore the data on the target.
  b. Run **`onbar -r -e -p`**.
  c. Establish communications between both database servers by running **`onmode -d RSS primary_server`**.
  d. Verify the success of the operation by checking the log file on both sides. The RSS server must be flagged as operational. You can run **`onstat`** on the RSS side. The new RSS server status should appear.

Using the ontape backup utility

Ontape provides you a simple backup solution that backs up the entire primary database server serially into a single file. The ontape utility should only be used if there is no onbar environment available and the database server backup is running in an appropriate time frame.

To set up the RSS server using ontape, complete these steps:

- On the primary database server:
  a. Back up the database server by running **`ontape -s -L 0.`**
  b. Copy the backup file produced by ontape to the target machine
  c. Register the new server by running **`onmode -d add RSS secserver`**.
- On the secondary:
  a. Run **`ontape -p`** to restore the backup.
  b. Establish communications by running **`onmode -d RSS primary`**.

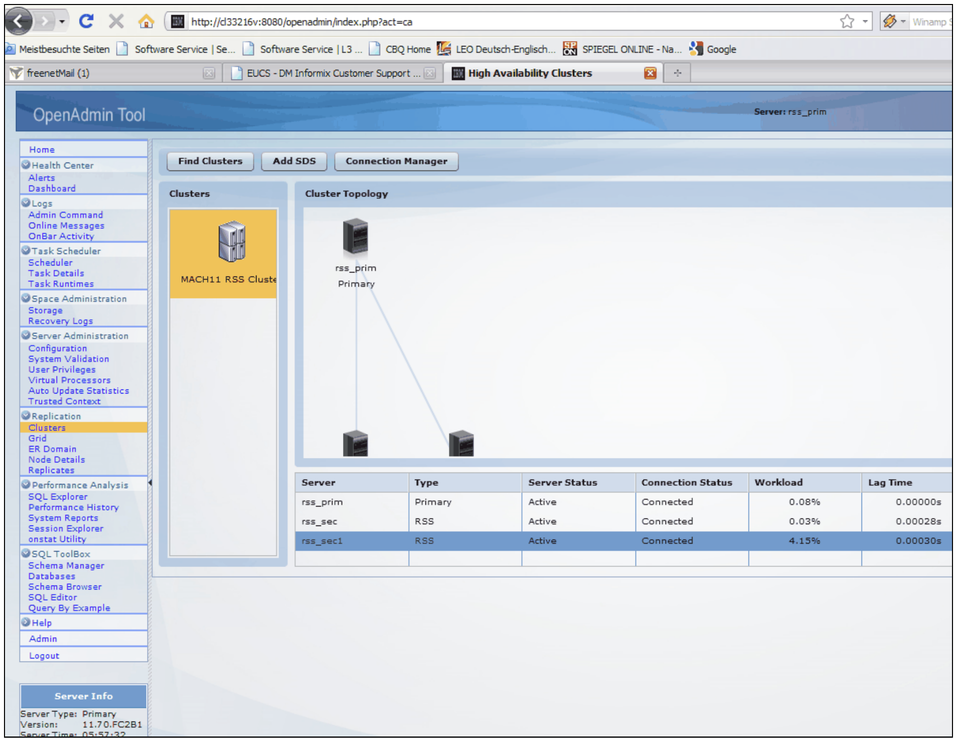
9.2.3 Verification of the setup

You can verify the success of the expansion of your RSS cluster using various approaches.

Using the OpenAdmin Tool

A simply verification method is to use the OpenAdmin Tool (OAT) administration console.

If the primary server has already registered the OAT, you can connect to the new RSS server from the primary by logging in to the primary server first, and then selecting **Replication** → **Cluster**. Click **Find Cluster**. In the cluster topology graphics, the new RSS server should appear, as shown in Figure 9-3.



Server	Type	Server Status	Connection Status	Workload	Lag Time
rss_prim	Primary	Active	Connected	0.08%	0.00000s
rss_sec	RSS	Active	Connected	0.03%	0.00028s
rss_sec1	RSS	Active	Connected	4.15%	0.00030s

Figure 9-3 Visualization of an existing RSS cluster in the OAT

If you are managing a new cluster within the OAT, add the connection of the primary server first. In the OAT main window, select **Admin** → **Manage Connections**. In next window, click **Add Connections**. Enter the connection parameters for your primary database server used in the RSS cluster. After registering the primary in the OAT, connect to the primary server and use the procedure described in the last paragraph to find the new RSS server.

Using the sysmaster database

To verify the status of the new RSS server using the sysmaster database, run the following command:

```
SELECT * FROM sysmaster:syssrcrss
```

The query output should show a row for the newly added server.

Using the onstat utility

To use the command-line monitoring interface, you can run **onstat** on the new RSS secondary server to check the status of the database server. The output of **onstat** should be as follows:

```
#Standard RSS
IBM Informix IDS Version 11.70.FC2 -- Read-Only(RSS) -- Up 00:41:45
#Updatable RSS
IBM Informix IDS Version 11.70.FC2 -- Updatable (RSS) -- Up 00:01:16
```

On the primary server, you can run **onstat -g rss**. The new RSS should be listed in the output. Example 9-1 shows a ample output.

Example 9-1 onstat -g rss on the primary server

```
Local server type: Primary
Index page logging status: Enabled
Index page logging was enabled at: 2011/02/21 04:14:02
Number of RSS servers: 2

RSS Server information:

RSS Srv      RSS Srv      Connection     Next LPG to send        Supports
name         status       status           (log id,page)         Proxy Writes
rss_sec      Defined      Disconnected           0,0             N
rss_sec1     Defined      Disconnected           0,0             N
```

In addition to using the **onstat** command, you can check the `online.log` file on the secondary server. Look for the following status line:

```
04:19:01  DR: RSS secondary server operational
```

Not successful

If an error occurs while switching the target database server to an RSS server, the target server remains in the fast recovery mode. If your RSS server does not show up or remains in the fast recovery mode, check the following items:

- Have you registered the new RSS server on the primary by running the **onmode -d**?
- Can both host machines talk with each other?

- Do you have the RSS and the primary added to the `sqlhosts` file?
- Did you run **`onmode -d`** on the RSS server to connect to the primary?
- Are both machines trusted for the informix user?

9.3 Updatable secondary versus read only standby

After the new RSS server is successfully established, all new changes applied on the primary are replicated continuously to the new secondary. The server now also accepts connects from new incoming database client applications. By default, only read-only SQL statements such as SELECTS are allowed for execution on an RSS server. Any attempts to change the data return the following SQL error:

```
insert into customer (customer_num , fname, lname )
values ( 0, "new_customer_fname","new_customer_lname")

   271: Could not insert new row into the table.
 ISAM error: operation illegal on a DR Secondary
```

The **`onconfig`** parameter UPDATABLE_SECONDARY defines the behavior of the secondary server. With a default setting of 0, the secondary server behaves as a read-only server. Any setting higher than 0 enables the RSS server to internally redirect any SQL based write operations, such as Update, Insert, or Delete statements, to the primary server. The SQL operation is applied there.

The value of UPDATABLE_SECONDARY can influence the general performance of DML statements of the local server. This parameter defines the amount of service threads created automatically when the secondary server starts. The service threads maintain the incoming SQL request sessions independently. If you expect a higher amount of incoming sessions running DML SQL statements on your secondary server, adjust UPDATABLE_SECONDARY to a higher number. Any DDL statements on an RSS server are not allowed.

9.4 onconfig parameters that affect RSS servers

The UPDATABLE_SECONDARY parameter is one major parameter that affects the secondary server's behavior. Other **onconfig** parameters that affect RSS servers are:

- TEMPTAB_NOLOG

  This parameter affects the default logging behavior of a temporary table created by the CREATE TEMP TABLE and SELECT INTO TEMP statements. The default setting of 0 creates tables with logging by default. A setting of 1 creates all temporary tables as non-logging tables in the temp dbspaces. Set TEMPTAB_NOLOG to 1 for an RSS server.

- DELAY_APPLY

  Setting this parameter might introduce a delay factor for rolling forward the incoming logical log information on an RSS server. The setting can be a number of days, hours, minutes, or seconds.

- STOP_APPLY

  Setting this parameter to 1 stops the apply of incoming logical logs immediately. To resume log apply, set this parameter to 0. Set this parameter to a time stamp to stop log apply at a specific time on the RSS server. Setting STOP_APPLYA dynamically using **onmode -wm** is more practical.

- LOG_STAGING_DIR

  You must set this parameter in case either DELAY_APPLY or STOP_APPLY is set on the RSS server and all the incoming logical log files cannot roll forward immediately. The log files have to be kept on disk for a later apply. This parameter points to a directory where the RSS saves the logs for later reply.

  The server creates a subdirectory in the specified directory. The name of the directory is listed in the `online.log` file when an RSS server starts. Each log file is saved in a separate file with an unique number postfix.

- FAILOVER_TX_TIMEOUT

  Use the FAILOVER_TX_TIMEOUT configuration parameter to enable transactions to complete after failover of the primary server.

A common use case scenario for the STOP_APPLY or DELAY_APPLY parameter is the execution of an external backup on the RSS server. Running an external backup on any primary database server requires a checkpoint followed by a block of any write operations. You can move the external backup from the primary to one of the RSS servers. To ensure a entire block is received, you must specify a STOP_APPLY. The data on disk is consistent and there are no ongoing changes applied until the server resumes.

DELAY_APPLY can be used if there are administration operations taking place on the production database server. With DELAY_APPLY set, the RSS server remains unchanged for a period of time. In case the database changes on the primary server failed, the secondary server can be used to reinstate the primary server to the before-change state. It is much faster than recovery through a complete restore.

9.5 Monitoring the RSS database server pairs

The SQL-based and command line-based interfaces let you manage the RSS cluster from an embedded or remote environment. Informix provides the following methods to check the RSS server's status:

- The OAT administration console
- The sysmaster database
- Event alarms
- The `onstat` command

OAT

OAT provides a combination of a visual depiction and a table oriented listing of the status and utilization of all configured database servers in the cluster, summarized at one place. You only have to add the primary database server to the connection database maintained by OAT. Thereafter, after you connect to the primary server from OAT, information about the whole RSS cluster becomes available.

Monitoring is not the only benefit of using OAT; you can also maintain the server and the Connection Manager at one place without locally connecting to the machine where the database server resides.

The sysmaster database

When you use the sysmaster database, all the information about the RSS servers becomes available to the remote clients or the SQL based applications. The following tables keep status information about the primary and the RSS servers in the MACH11 cluster:

- syssrcrss
- systrgrss
- sysrsslog

More information about the connection protocol (known as Server Multiplexer Group (SMX)) between the pair can be obtained by querying the syssmx table.

You can obtain the column description of these three tables from the schema file of the sysmaster database located in `$INFORMIXDIR/etc/sysmaster.sql`.

You also can use **`onstat -g rss`** to obtain information about the RSS servers.

Event alarms

The database server uses the event alarm feature to report a situation that requires the immediate attention of a database administrator. To use the event alarm feature, set the ALARMPROGRAM configuration parameter to the full path name of an executable file that performs the necessary administrative actions. A simple example would be a shell script that informs the database administrator about the event either through email or a pager.

The database administrator can also monitor the event alarms that are triggered for the RSS servers. When each event alarm is triggered, the primary server writes a message to the message log that includes the RSS server name. There are several situations when the primary server triggers event alarms for the RSS server.

These are the most important alarms that need to be monitored:

- `RSS log replay position is falling too far behind the RSS source.`

  This message displays when the log replay position for the RSS server is falling too far behind the primary. If this trend continues, the primary might not have a logical log file available when the RSS server requests it.

- `RSS server is not acknowledging log transmissions.`

  If the primary server does not get acknowledgement (ACK) back (after it reaches the limit of successful transmissions) due to a network issue, it will trigger this event alarm. The primary server will not send any more logical log pages until an acknowledgement is received.

- `Error receiving a buffer from RSS server.`

  The primary server experienced an error receiving a message from the RSS server.

The class ID used by these events is 40, and the severity is ALRM_ATTENTION.

The onstat interface

Informix provides several options for the onstat utility to obtain various details about the cluster's status:

- **`onstat -g rss`**
- **`onstat -g rss <secondary_server_name>`**
- **`onstat -g rss verbose`**

Depending on where you run **`onstat`**, the output may be different depending on whether the local server is the primary or the secondary. The output matches the content of the syssrcrss and systrgrss tables in the sysmaster database.

To obtain more detailed information about the connection layer supporting RSS communication, run the following commands:

- **`onstat -g smx`**
- **`onstat -g smx ses`**

The output of a specific onstat utility option matches the content in the sysmasters syssmx database table.

9.6 Changing modes

One of the major reasons to use an RSS server in a cluster environment is to ensure data availability. If there is a system failure, whether it is a storage failure or a complete loss of the data center, you need to be able to run a successful failover to ensure the continued access to and maintenance of your data. In this section, we introduce certain server switch possibilities for an RSS server.

The onmode utility is used to change the modes of a server.

From RSS to HDR secondary

Let us assume that you have a environment using HDR and RSS in parallel. If for any reason the HDR secondary server fails and it appears that the HDR secondary server will be offline for an extended period of time, then the RSS server can be converted into the HDR secondary server by issuing the following command:

```
$ onmode -d secondary primary_servername
```

When the RSS server is converted to an HDR secondary server, the primary server deletes the RSS information first and then the RSS server is converted to an HDR secondary server.

If an HDR secondary server already exists, an error is generated and a message saying that an HDR secondary already exists will be put in the online log.

From HDR secondary to RSS

If the HDR secondary is not able to keep up with the primary server due to network latency, and the primary server's performance is affected, then for that transient period the HDR secondary can be converted to an RSS server by using the following command:

```
$ onmode -d RSS primary_servername
```

Later, when network bandwidth is restored, the RSS server can be converted back to an HDR secondary server.

From RSS to standard server

If at any time an RSS server needs to be taken out of the high availability cluster, then the RSS server should be converted to a standard server by issuing the following command:

```
$ onmode -d standard
```

This command the RSS server a stand-alone server, and it will no longer be part of a cluster. To move this stand-alone server back into the cluster as an RSS server, complete the following steps:

1. Restore the Level 0 archive of the primary server.
2. Add the RSS server information to the primary.
3. Issue **onmode** to set up the RSS server.

10

Shared Disk Secondary servers

In contrast to the RSS and HDR architectures, the Shared Disk Secondary (SDS) architecture provides the ability to set up multiple database servers sharing the whole dbspace set defined by a primary database server.

The benefits of this feature in terms of resources, compared to RSS, include significantly lower requirements for disk space and a slight reduction in network traffic. The ability to add and remove resources dynamically, easily, and quickly make the SDS cluster environment a solid base for database cloud solutions. Adding new database server resources temporarily to an existing network that serves season and quarterly peak business requirements in combination with a smart client distribution is the key to effective resource utilization and application load balancing.

The SDS functionality can be easily integrated into heterogeneous IBM Informix database cluster and cloud solutions, such as RSS, HDR, ER, and Flexible Grid.

In this chapter, we discuss the following topics:

- Where to use SDS servers
- Setup and configuration of SDS servers
- Monitoring and troubleshooting SDS servers
- Failover considerations for SDS servers

10.1 Basic concepts

In this section, we start with a description of certain basic concepts of an SDS implementation. This information will enable a better understanding of the meaning of the parameters that must be specified in the SDS configuration and their expected values.

10.1.1 Dbspaces and load balancing

Unlike HDR secondary and Remote Secondary Standby (RSS) servers, an SDS server does not maintain a copy of the database server defined dbspaces on its own disk; rather, it shares the same physical disk with the primary database server. This setup means you can either define the SDS environment with a different server on one machine or on multiple machines that are using shared file systems to enable them to see the dbspaces defined by the SDS primary database server. Similar to the RSS feature, multiple secondary servers can be attached to the SDS infrastructure. In addition, secondary servers can be attached and detached on demand depending on the actual given hardware infrastructure and the workload.

SDS servers provide increased availability and scalability without needing to maintain multiple copies of the physical layout, which results in lowering data storage costs.

Figure 10-1 shows a simple un-nested SDS infrastructure with three secondary nodes.

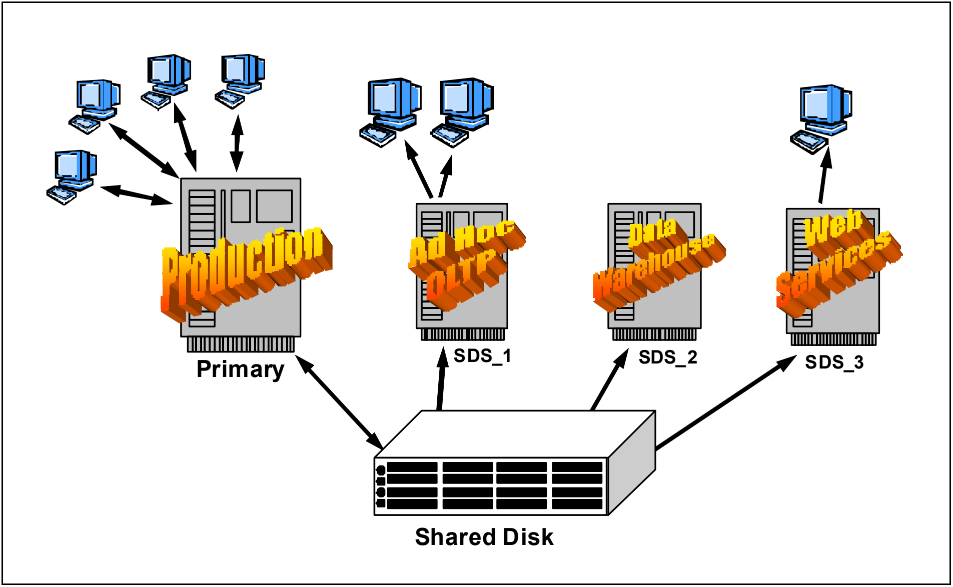


Figure 10-1   SDS cluster environment for smart client application workload distribution

By allowing one or more instances of the Informix database server to attach to same shared disk subsystem, SDS servers provide redundancy for the server in addition to data redundancy solutions. Multiple SDS servers also provide the opportunity to dedicate specific SDS servers for specific tasks, such as data warehousing on a DSS oriented server or web application server with an OLTP approach with the appropriate differences in the configuration for parallel data query (PDQ) and memory requirements. The SDS environment can also be used simply for work balancing by spreading the existing company applications across the SDS servers in the infrastructure to achieve a better throughput.

10.2 Configuration and setup

Expanding an existing SDS cluster or creating a new SDS cluster is easy and quick. Because all the servers share the same storage, there is no need for backup and restore or cloning of the new server. The setup requires only a few changes in the configuration files on both sides of the new database server pair. In the following section, we describe the settings required to attach a new SDS server to a cluster.

10.2.1 Primary database server

In a shared disk cluster environment, the primary database server is the owner of a disk and has read and write access. Complete these steps to prepare the primary server for the SDS server to be added to the cluster:

1. In the `sqlhosts` file:
   a. Specify at least one network based connection protocol for the primary SDS server when creating a new cluster.
   b. Add the communication parameters of all the new SDS database servers to enable successful replication.
2. Announce the primary server if you are setting up a new SDS cluster. Run the following command:

```
onmode -d set SDS primary <dbservername of the primary>
```

10.2.2 Secondary server

The SDS server uses the definitions of the dbspaces from the primary, so there is no need to create “cooked” files or configure raw devices for normal dbspaces, blobspaces, or sbspaces. There are only a few changes required in the new server.

sqlhosts file changes

You have to create an `sqlhosts` file that contains the entries for the secondary and the primary SDS servers. Both server definitions have to be based on a network communication protocol. If the secondary is defined on the same server as the primary, you could share the `sqlhosts` file of the primary SDS server and add the new SDS node there. If there is a separate machine for both servers, you also have to add the newly defined SDS secondary into the `sqlhosts` file on the SDS primary.

Defining the SDS server in a server group is also supported and should be considered for application failover in case the primary connection server of a particular application environment becomes temporarily unavailable.

Changing the onconfig parameters

Set the following **`onconfig`** parameters on the SDS secondary server:

- SDS_ENABLE

  This parameter defines the new database server as an SDS server. Set the value to 1.

- SDS_PAGING

  The content of logical logs generated by the primary is continuously applied on the SDS server. Changes of data in the roll-forward make memory buffers dirty until the cleanup in the next checkpoint. In the case of a parallel workload on the secondary server with connected user sessions, it is possible that a dirtied page has to be flushed out, triggered by a LRU_MAX_DIRTY event or by a foreground write. Any dirty buffers on an SDS will not be written to disk, because this task can only be done by the primary server. All these buffers are written to a file referred to as a page file.

  There are two page files that must be defined. These files are alternately used from checkpoint to checkpoint.

  The expected format for the value of the parameter is SDS_PAGING <path>,<path1>.

- SDS_TEMPDBS

  Specific select statements, including those executed on a read-only server, require the existence of a temporary dbspace. You are allowed to create temporary tables or read data into a temporary tables for further use. Additionally, you need temp space to materialize views or table expressions, writing overflow partitions for hash join queries, or to do sorting. In an SDS cluster environment, the secondary is not able to share the temporary dbspaces defined by the primary for writing. This restriction requires that the secondary server has to specify its own temporary dbspace, which is visible only in scope of the secondary.

  You have to specify the following values for each temporary dbspace for the parameter:

  – Name of the dbspace
  – File name, including the path
  – Pagesize
  – Offset in the file
  – Size of the dbspace chunk file

  You may have more than one temporary dbspace specified in the `onconfig` file of an SDS.

Example 10-1 shows sample settings for these parameters.

Example 10-1 Sample settings for SDS specific onconfig parameters

```
SDS_PAGING      /dbspaces/sds11_sec/pagefile,/dbspaces/sds11_sec/pagefile1
SDS_TEMPDBS tempspace,/dbspaces/sds11_sec_2K/sds_sec_tempspace,2,0,10000
SDS_TEMPDBS tempspace1,/dbspaces/sds11_sec_4K/sds_sec_tempspace1,4,0,50000
SDS_TEMPDBS tempspace2,/dbspaces/sds11_sec_8K/sds_sec_tempspace2,8,1000,100000
```

Starting the secondary server

After applying the changes to the `onconfig` file, start the SDS secondary by running $**`oninit`**.

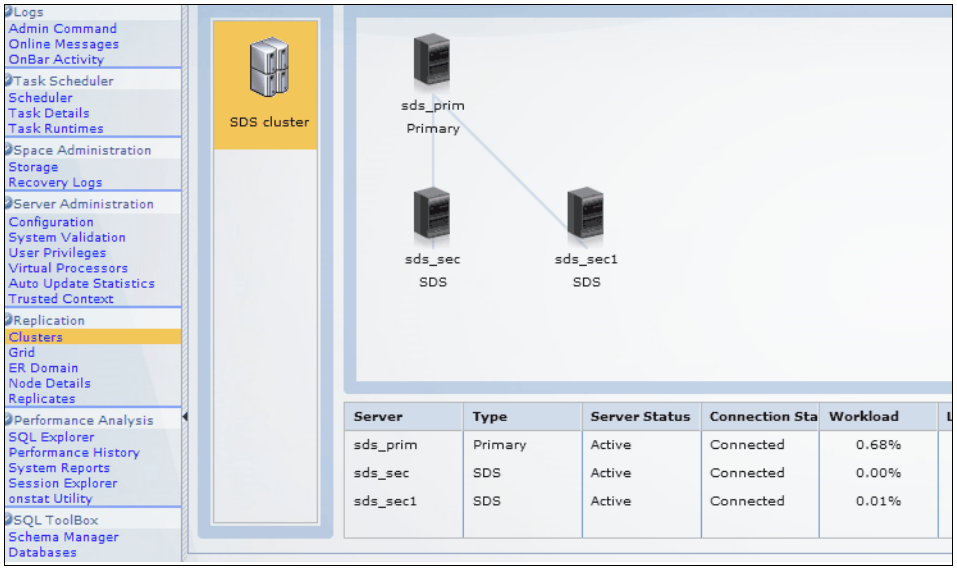
10.2.3 Verification of the setup

You can verify the success of the expansion operation of your SDS cluster in many different ways.

Using OAT

A simply way to verify the setup is use the OAT administration console. If the primary server has been registered in the OAT, select **Login** and connect to the primary server. Select **Replication** → **Clusters**. Click **Find Clusters**. The new SDS server should appear in the cluster topology graphics.

Figure 10-2 is an example of how the OAT window shows the MACH11 cluster.



Server	Type	Server Status	Connection Sta	Workload
sds_prim	Primary	Active	Connected	0.68%
sds_sec	SDS	Active	Connected	0.00%
sds_sec1	SDS	Active	Connected	0.01%

Figure 10-2 Visualization of the currently defined SDS cluster environment using OAT

To manage a new cluster with OAT, register the primary server first. Launch OAT and select **Admin** → **Manage Connections**. In next window, click **Add Connections** and enter the connection information of the primary server of the SDS cluster. After registering the primary in OAT, connect to the primary server and discover the SDS secondary servers using the steps described above.

Using the sysmaster database

To verify the status of the new SDS server using the sysmaster database, run the following SELECT statement on the primary database server:

```
SELECT * from sysmaster:syssrcsds
```

The query should show a row for each newly added SDS database server.

Using the onstat utility

You can run **`onstat`** on the new SDS server to check the database server's status. Here is an example output:

```
#Standard SDS
IBM Informix IDS Version 11.70.FC2 -- Read-Only (SDS) -- Up 00:56:21
#Updatable SDS
IBM Informix IDS Version 11.70.FC2 -- Updatable (SDS) -- Up 01:18:22
```

On the primary server, you can run **onstat -g sds**. The new SDS should be listed in the output, as shown in Example 10-2.

Example 10-2 onstat -g sds on the primary server

```
Local server type: Primary
Number of SDS servers:2

SDS server information

SDS srv      SDS srv      Connection       Last LPG sent       Supports
name         status       status           (log id,page)       Proxy Writes
sds_sec1     Active       Connected              3,2474        Y
sds_sec      Active       Connected              3,2474        N
```

Using the online log

In addition to the **onstat** command, you can also check the `online.log` file on the secondary server for verification. Look for the following status line:

```
04:19:01  DR: SDS secondary server operational
```

10.3 Troubleshooting the setup phase

There are several environment settings that can cause unexpected results during the initialization and setup phase of a new SDS. Most of the time, the problems come from the following areas:

- There is a missing or incorrect network definition parameter in the `sqlhosts` file on one of the sides. Possible solutions are:
  - Check that the specified network protocols are supported.
  - Check that the communication parameters, such as the host name and port numbers, are correct.
  - Check if there are any missing, incomplete, or incorrectly defined **onconfig** parameters.
  - Check the parameters, especially SDS_TEMPDBS and SDS_PAGEFILE, for correct spelling of the values.
  - Check if SDS_ENABLE is set to 1.
- There are missing files or directories that have to exist before starting the SDS server for the first time. Possible solutions are:
  - Make sure that the path of the page file exists and has the appropriate permissions.
  - Make sure that the files specified for the temp dbspaces on the SDS exist.

To diagnose the errors during the initialization of an SDS, check the messages at the terminal where **oninit** was executed or the `online.log` file. Most of the time, the error is indicated at one of these two places. Wrong configuration parameter settings are declined with a message in the start window. Communication problems are reported in the log file of the database server.

10.4 Updatable secondary servers

In a default configuration, SDS servers allow only the read-only access to their maintained databases. You are able to run the SELECT statements successfully against the data in your database server. DML statements such as INSERT, UPDATE, and DELETE return the following error:

```
insert into customer (customer_num , fname, lname )
values ( 0, "new_customer_fname","new_customer_lname")

   271: Could not insert new row into the table.
 ISAM error: operation illegal on a DR Secondary
```

The **onconfig** parameter UPDATABLE_SECONDARY defines the behavior of the secondary server. With the default setting of 0, the secondary server behaves as a read-only server. Any setting higher than 0 enables the SDS to internally redirect any SQL-based write operations such as Insert, Update, and Delete statements, to the primary server, where they will be applied. The value of the parameter can influence the general performance of the DML statements of the local server. It defines the number of service threads created automatically when the secondary server starts. The threads maintain the incoming SQL request sessions independently. In case you expect a higher amount of incoming sessions running DML SQL statements on your secondary server, adjust this value to a higher number. Any DDL statements on the SDS server are not allowed.

10.5 The onconfig parameters that affect SDS servers

We have discussed the major **`onconfig`** parameters required for the initialization of a new SDS server in the MACH11 cluster environment. There are a few more parameters that affect the behavior of SDS servers:

- SDS_TIMEOUT

  SDS_TIMEOUT specifies the amount of time in seconds that the primary server will wait for a log position acknowledgement to be sent from the SDS database server. If there is no log position acknowledgement received from the SDS secondary server within the specified time, then the primary server will be disconnected from the SDS server and continue.

- LOG_INDEX_BUILDS

  Enables the primary server to log all index operations in the logical log file, to replicate the changes to the secondary. This parameter can be set but is not required on SDS.

- TEMPTAB_NOLOG

  The parameter affects the handling of the default logging behavior of the temporary tables created by CREATE TEMP TABLE and SELECT INTO TEMP. The default setting 0 defines that the tables are created with logging by default. With a setting of 1, all the temporary tables are created as non-logging tables in the temporary dbspaces maintained locally. Set TEMPTAB_NOLOG to 1 for an SDS server.

- FAILOVER_TX_TIMEOUT

  Use the FAILOVER_TX_TIMEOUT configuration parameter to enable transactions to complete after failover of the primary server. This is valid only for an updatable secondary server.

- FAILOVER_CALLBACK

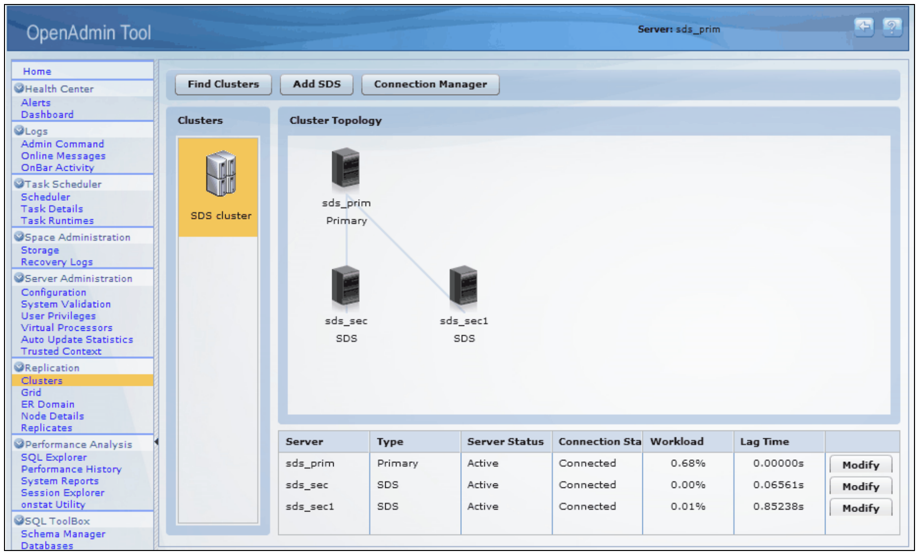
  This parameter defines a script or executable to be called if an SDS server is promoted to a primary server.

10.6 Maintaining SDS cluster environments using OAT

OAT provides the ability to maintain cluster environments. You can discover existing clusters, display their topology graphically, and manage the status of the associated servers in the cluster. Additional SDS servers can be started flexibly. Unused database resources can be easily detached and shut down.

As a demonstration of how easy it is to use OAT to maintain SDS clusters, we provide the following demonstration of adding a new SDS to an existing cluster:

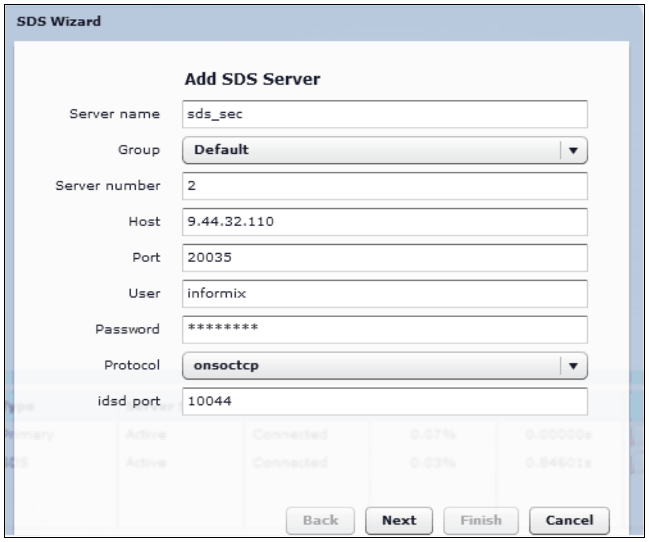
1. Launch OAT, select **Login** → **Replicate** → **Cluster**, click **Add SDS**. In the Add SDS Server window (Figure 10-3), specify the required connection parameters for your new server and click **Next**.



Server	Type	Server Status	Connection Sta	Workload	Lag Time	
sds_prim	Primary	Active	Connected	0.68%	0.00000s	Modify
sds_sec	SDS	Active	Connected	0.00%	0.06561s	Modify
sds_sec1	SDS	Active	Connected	0.01%	0.85238s	Modify

Figure 10-3 Connection parameter for a new SDS database server in the cluster

2. Specify the required configuration files, as shown in Figure 10-4.

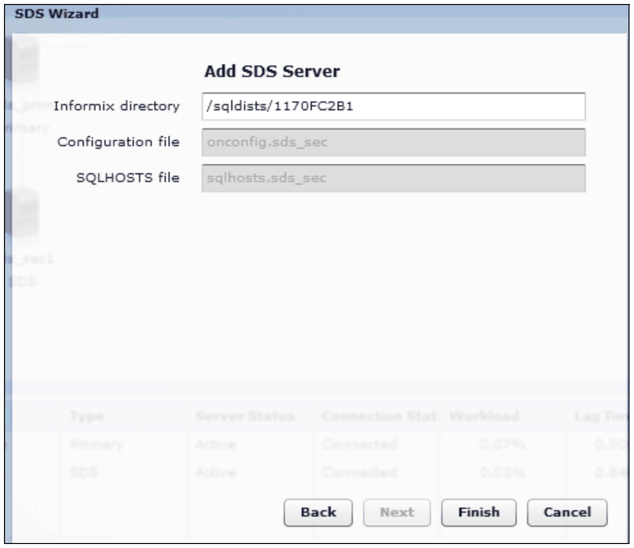

SDS Wizard

Add SDS Server

Server name	sds_sec
Group	Default
Server number	2
Host	9.44.32.110
Port	20035
User	informix
Password	********
Protocol	onsoctcp
idsd port	10044

Figure 10-4 Configuration files for a new SDS server in the cluster

3. The new server should appear in the topology graphics, as shown in Figure 10-5.


SDS Wizard

Add SDS Server

Informix directory	/sqldists/1170FC2B1
Configuration file	onconfig.sds_sec
SQLHOSTS file	sqlhosts.sds_sec

Figure 10-5 New SDS server added to an existing SDS cluster environment

To use the functionality of the OAT, there must be a service daemon established on the target server machine that executes remote start and shutdown requests. Complete these steps to set the required services:

1. Make sure that you have the IBM Security Kit installed and create a certification label for the idsd daemon by running the following commands:

   ```
   gsk7cmd -keydb -create -db <mydb.kdb> -pw <mypassw0rd> -type cms -stash
   gsk7cmd -cert -create -db <mydb.kdb> -pw <mypassw0rd> -label
   <mycertificatelabel> -dn <"CN=lenexa.ibm.com,O=ibm,OU=IBM HTTP
   Server,ST=KS,C=US"> -size 1024 -default_cert yes
   ```

2. Set up the xinetd daemon to use idsd on UNIX. Edit the `/etc/xinetd.conf` file and add the following section:

   ```
   service idsd
                {
                disable = no
                socket_type = stream
                protocol = tcp
                user = root
                wait = no
                server = </sqldists/11.70.FC2/bin/idsd>
                server_args = -l </tmp/idsd.log> -k
   </sqldists/11.70.FC2/tmp/mydb> -n <mycertificatelabel>
                }
   ```

3. Restart the xinetd daemon on your system.

For more information about OAT, see Informix Information Center at the following address

http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp?topic=/com.ibm.oat.doc/ids_oat_059.htm

10.7 Monitoring SDS cluster environments

After setting up the SDS environment successfully, there are several ways to monitor the status of the associated SDS servers:

- The OAT graphical administration console
- The sysmaster database
- The onstat command-line executable

Similar to most of the other server features, the onstat utility provides several options to see the status of the SDS servers. In addition, the sysmaster database contains some new tables that can be accessed using the SELECT statements for further processing in a open source or user defined web-based administration console.

10.7.1 The OAT administration console

OAT has the ability of cluster discovery for all database servers participating in any Informix database server cluster environment. After the primary database server is registered, the topology of the cluster and the status of all participants can be easily obtained. The status of the participating SDS can be changed, meaning an existing SDS database server can be started and shut down. Additionally, new SDS servers can be established in your cluster using the OAT application.

Figure 10-2 on page 401 shows the entry window of the cluster management of OAT. For additional information about server management in a cluster environment, refer to 10.6, “Maintaining SDS cluster environments using OAT” on page 404.

10.7.2 The sysmaster database

When you use the sysmaster database, all the information about the SDS server becomes available to remote clients or SQL-based applications. The following tables have status information about the primary and the SDS servers in the MACH11 cluster:

- syssrcsds
- systrgsds

You can obtain information about the connection protocol between the pair of servers by querying the syssmx table.

The schema file of the sysmaster database `$INFORMIXDIR/etc/sysmaster.sql` contains the column description of syssrcsds and systrgsds.

You can also use **`onstat -g sds`** to obtain the server information.

10.7.3 Monitoring SDS with onstat

The onstat utility provides a set of options for monitoring SDS servers. Running **`onstat -g sds`** on the primary server and the secondary server can produce different result.

Consider using the following **`onstat`** options:

- **`onstat -g sds`**

  When you use this command, you can see an advance in the log apply position, which verifies a healthy communication between the SDS servers in you cluster environment.

  - On the primary database server, this command gives you the list of currently attached SDS servers. You also can monitor the communication flow between the servers by viewing the logical log positions on two servers.
  - On the secondary server, this command gives you the connection status and the actual log position received from the primary and applied to the data locally.

- **`onstat -g sds verbose`**

  The most important information you want from this command's output is the page file utilization on the SDS server. If the page file size is large, consider increasing the buffer pool size.

10.8 Special considerations

In this section, we discuss considerations for the administration of an SDS database server cluster environment.

10.8.1 Restarting the primary SDS server

When the SDS primary server is shut down, the currently available SDS secondary servers remain online and stay in read-only mode. When the SDS primary restarts, all the secondary servers are shut down automatically and have to be restarted manually. an SDS secondary server has to be switched to become the primary.

If an updatable secondary SDS server is used and the primary server is brought down, any write operations return an error. The SDS automatically switches to the read-only mode. The server also goes down when the SDS primary server is restarted.

10.8.2 Backup and restore

Only the SDS primary server supports a backup using **`ontape`** and **`onbar`**. A backup on an SDS secondary return the message `Server is in an incompatible state`.

A subsequent restore of the primary SDS database server will behave similar to a restart of the server in view of the SDS secondary servers. Immediately after the primary server moves to the recovery mode, it sends a termination request to the attached secondary servers, which will cause them to shut down.

10.8.3 Changing modes

Adding and removing an SDS server to and from the current cluster is flexible. Therefore, an SDS database server can be switched to a primary or secondary server quickly and easily.

Moving an SDS primary to a standard server

You can switch an SDS primary server to be a standard server by running **`onmode -d clear SDS primary <primary servername>`**. Using this particular command, the transition can only be performed when the SDS primary server is running in a stand-alone mode, meaning that there is no SDS secondary servers up and running. To change an SDS primary with an attached SDS secondary running immediately into a stand-alone server, use the -force option of the **`onmode -d`** command. This option causes the secondary server to stop and the primary to move into stand-alone server mode.

Moving an SDS secondary to an SDS primary server

Any SDS secondary server can be promoted to be the primary in the cluster. This functionality allows a flexible switch of master responsibilities in case of an outage of the original primary server. Use this command to fail over an SDS primary server to an SDS secondary server:

```
onmode -d set SDS primary <new primary servername>
```

If the old primary was not down, the server will be shut down automatically. All other attached secondary SDS servers will automatically switch to the new primary server in the cluster.

10.8.4 SDS and other HA solutions

Sharing disks and reducing network traffic are the strengths of SDS and differentiate this solution from the other Informix database cluster environments. However, do not limit SDS to be only a stand-alone workload balancing solution. It can easily be plugged into any existing ER or Flexible Grid infrastructure with the database servers that use different versions. Also, the combination of RSS and SDS allows you to take the advantages of the strength of both solutions: availability, replication, and client distribution. Any SDS primary can also serve as an RSS primary and HDR primary, or as an ER source or target system. We discuss the variety of combinations of the Informix high availability solutions in more detail in Chapter 7, “MACH11 clusters” on page 309.

11

Connection management in an Informix environment

The IBM Informix database server offers several high availability (HA) options. The MACH 11 cluster provides well defined failover management between the participating database servers if there is a server outage. Flexible Grid lets you replicate across servers with disparate environments with minimal administration requirements. Customers may have any or a combination of these HA options in place, have multiple clusters with one cluster defined for each region, or have a cluster and a grid environment with one of the nodes in the grid participating in the cluster.

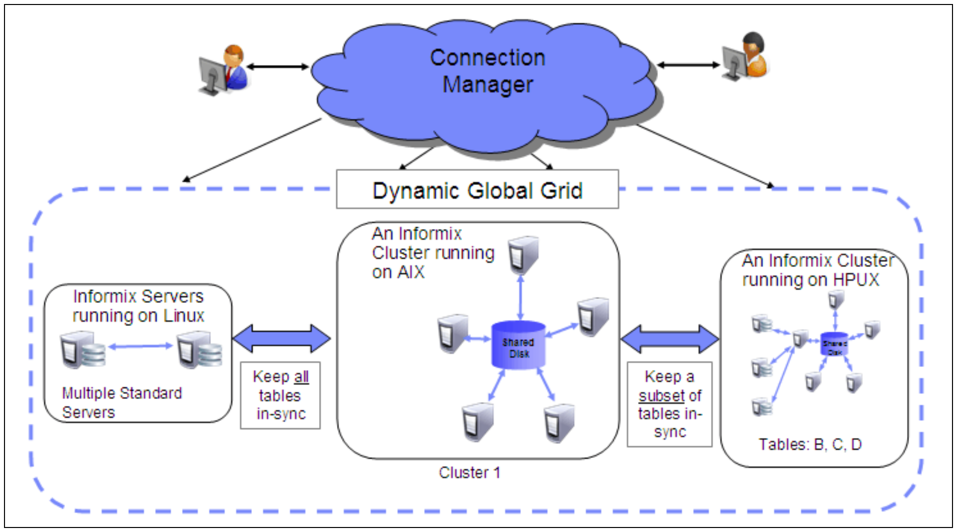
IBM Informix Connection Manager is a thin and lightweight layer between the client and the database side setup that shears the complex topology from the client. The Connection Manager acts as a delegate to route the client connections to the appropriate server based on service level agreements (SLA), and it is also the failover controller and coordinator.

In this chapter, we provide a detailed description of the capabilities of the Connection Manager. The major focus points are:

- Configuration and setup
- Client considerations
- Monitoring and troubleshooting
- Common use case scenarios

11.1 Informix Connection Manager

Informix Connection Manager provides client workload partitioning and an automatic failover arbitration capability, based on a defined task acquisition strategy, in case the primary database server in the cluster becomes unavailable. The Connection Manager is distributed with Informix Client Software Development Kit (Client SDK) and can handle multiple grids and clusters in a single configuration. This is an enhancement over the previous version where each cluster or grid needed a dedicated CM. Complex environments can be handled with fewer Connection Managers as compared to the previous release. Figure 11-1 shows an example layout where the Connection Manager works with multiple grids and clusters



Connection Manager
Dynamic Global Grid
Informix Servers running on Linux
Multiple Standard Servers
Keep all tables in-sync
An Informix Cluster running on AIX
Shared Disk
Cluster 1
Keep a subset of tables in-sync
An Informix Cluster running on HPUX
Shared Disk
Tables: B, C, D

Figure 11-1 Connection Manager working with multiple grids and clusters

Client applications running in a database environment are classically divided into OLTP and DSS applications based on the transaction types. The reasoning here is to separate both kinds of applications because of their different requirements in terms of memory, CPU utilization, and response times. But can a DSS application be run with specific steps by generating intermediate aggregation tables on an HDR secondary read-only server? For shipping clients over a specific number of database servers available in an Informix cluster environment, this classification is only a rough breakdown. We need to have a finer granularity for categorizing database applications based on parameters, such as I/O requirements, CPU utilization, memory utilization, PDQ requirements or response time specifications, read-only (report generating) based applications, and so on.

The classification of the applications running in a production environment must be done either by the application development side or the database server administration side in your company. This is something that the Connection Manager cannot take over. But after you have identified the relationship between the application types and the database server maintained by the cluster, the Connection Manager can provide the complete transparent client distribution to achieve a fair balanced utilization of the database resources according to the guidelines.

You can bind applications to a certain group of servers organized by SLAs, such as SDS, SDS+ RSS, or SDS+RSS+Primary, to the primary database server, or to any database server that is part of a grid or replication set, or to a particular database server specified by its name. For example, if you specify an SDS server, the Connection Manager looks at the connection time of a new client, which should be tied to the SDS policy for which SDS servers are currently available, and then redirects the client to the most suitable of them. Because the Connection Manager relies on the same administration mechanism for incoming clients, in terms of connections, it takes little effort to use the tool to configure an existing Informix cluster environment. The only requirements are the specification of new connection aliases in the `sqlhosts` file, the definition of the routing of the clients and, in case you want to use the arbitrator capabilities also, the application of a failover strategy in the case of clusters.

11.1.1 Installation of the Connection Manager

The Connection Manager, offered as an utility named ONCMSM, and the dependent utilities, such as ONPASSWORD, are shipped with the IBM Informix Client SDK or Connect package. The current version of the Client SDK or Connect product is Version 3.70. This Client SDK package is either available in the Informix 11.70 database server bundle or is distributed as a stand-alone product.

ONCMSM is a connectivity utility and should be installed either on the machines where the database applications are running or on an intermediate machine separated from the cluster, grid, or other entities that it works with, to lower the administration impact.

11.1.2 Configuration of the Connection Manager on UNIX

After the Connection Manager utility ONCMSM is installed, you can configure the Connection Manager to meet your business requirements using the following configuration files and system variables:

- Environment variable settings
- `Sqlhosts` file
- Connection Manager configuration file

In the following section, we describe the appropriate settings of the different parameters and how they influence the operation of the ONCMSM utility.

Environment variables

The ONCMSM utility behaves similarly to other database applications. Therefore, the Connection Manager requires the setting of the following environment variables:

- INFORMIXDIR: Pointing to the Client SDK or Connect installation directory.
- INFORMIXSQLHOSTS: Containing the SLA port definitions for the clients and the Connection Manager to connect to the primary server in the cluster or one of the nodes in the grid, or all of the servers in the server set.

Changing the sqlhosts file

The primary database server entry of the appropriate Informix cluster and one of the server entries for the grid environment must be added to the particular `sqlhosts` file used by the Connection Manager. In addition, the Connection Manager requires the specification of entries for the Service Level Agreements (SLAs). In the next section, we explain why SLAs are required, as well as their relationship to the database server and the application they support.

Configuration files for ONCMSM

A configuration file is an input file that the Connection Manager reads for its setup. It lists the Connection Manager name registered with different servers, the log and debug parameters, and the service level agreements (rules) for the grid and cluster with which the Connection Manager has to monitor and work. The configuration file is passed to the Connection Manager binary ONCMSM as a parameter at start time (with the -c option), for example, **`oncmsm -c myconfig`**. The prior version of the Connection Manager allows either a configuration file or command-line options, whereas Version 3.7 enforces the use of a configuration file.

The Connection Manager configuration file has several sections. Example 11-1 shows a sample configuration file. There is global information in the beginning, followed by sections specifying particular settings. In this example, under the CLUSTER c1 section, we have entries for the primary Informix server, several SLAs, and failover criteria for that cluster.

Example 11-1 Sample configuration file

```
NAME            cm1connection manager
LOG             1
LOGFILE         ${INFORMIXDIR}/tmp/cm1.log

DEBUG   1
CLUSTER c1
{
        INFORMIXSERVER  ol_ids_1
        SLA     oltp1   DBSERVERS=primary
        SLA     myhdr   DBSERVERS=hdr
        SLA     myrss   DBSERVERS=RSS
        SLA     mysds   DBSERVERS=SDS
        FOC     ORDER=SDS,RSS TIMEOUT=5 RETRY=1
}
CLUSTER c2
{
        INFORMIXSERVER  ol_ids_2
        SLA     oltp2   DBSERVERS=primary
        FOC     ORDER=HDR,SDS,RSS TIMEOUT=5 RETRY=1
}
SERVERSET       sset1
{
        INFORMIXSERVER ol_ids_1,ol_ids_2
        SLA     demo1   DBSERVERS=(ol_ids_1,ol_ids_2)
}
```

The sections and the parameters in the configuration file are as follows:

- NAME: This parameter indicates the name for the Connection Manager. This information is what will show up in the **`onstat -g cmsm`** output. This is a mandatory parameter and has a global scope.
- LOG: This parameter sets whether the Connection Manager should log its activities. It goes with the LOGFILE parameter. A value of 1 means to store the basic activities This is an optional parameter and has a global scope.
- LOGFILE: This parameter points to the file where the log and debug information is stored. This is an optional parameter and has a global scope.

- DEBUG: This parameter enables ONCMSM to write more status information to the log file. The values are 1 and 0. Set DEBUG to 1 when troubleshooting to identify the root cause of the current problem and when the current status messages in the log file are not sufficient. Values are 1 and 0. This is an optional parameter and the scope is global.
- Types sections:

  The type names can be CLUSTER, GRID, REPLSET, or SERVSET. Each type section should have a name associated with it. This name will show up in the **`onstat -g cmsm`** output as SLA NAME, TYPE, or TYPE NAME. There has to be at least one type section in a configuration file

  The type section can have the following parameters. The parameter's scope applies to the section in which it is defined. They are:

  - INFORMIXSERVER: This entry points to the servers that are key for that section. It generally points to a primary server in the cluster, a server node in a grid, a server in the replicate set, or servers in a SERVSET type. This is a mandatory parameter within a section. If this entry has a number of servers, the Connection Manager uses the first available server in the list at the time of initialization.
  - SLA: This parameter defines the service level agreement. Each section has to have a mandatory SLA entry. The value of the SLA parameter contains the name of the SLA and a redirection policy. The redirection policy specifies the type or the name of the target database server to which the incoming client should be connected. The name of the SLA must be used by the SQL connect statement by the database clients, similar to a database server name. This statement allows the client to successfully connect to the Connection Manager.

    The SLA parameter appears multiple times in a section (and across sections) with different names. The number of SLAs defined in each section depends on the database server administrator, which means that the Connection Manager can handle multiple SLAs in parallel. Each SLA has a list of database servers that participate in fulfilling the SLA.

    Make sure that for each defined SLA represented by the name, an appropriate entry in the `sqlhosts` file used by the Connection Manager exists. At start time, the ONCMSM reads the `sqlhosts` file. In addition to reading the connection information for the primary database or one of the grid node servers, the SLA related entries are also extracted to investigate the available servers for the appropriate cluster and grid.

For each SLA, a so-called *listener thread* is installed at the specified port on the local machine. This thread serves the incoming client requests. The ONCMSM connects them according to the given policy of the appropriate database server.

SLA has the following syntax:

```
SLA <name> <attribute>=<value>...
```

Where

<name>	The name for the SLA. This is mandatory for an SLA entry.
<attribute>	One of the following:

- DBSERVERS: An explicit server name or the server type to which the client application connects. It can be a MACRO as well. This is mandatory for an SLA entry.

  The DBSERVER values can be a server type or an explicitly named server. Table 11-1 list the possible values of DBSERVER.

  Table 11-1 DBSERVER values

Value	Description
Primary	The client will be redirected to the primary database server in the cluster.
SDS	The client will be redirected to an SDS secondary database server. If multiple SDS servers are available, clients are allocated across them.
SDS	The client will be redirected to an SDS secondary database server. If multiple SDS servers are available, clients are allocated across them.
RSS	The client will be redirected to an RSS secondary database server. If multiple RSS server are available, clients are allocated across them.
HDR	The client will be redirected to an HDR secondary database server.
ANY	This value identifies any server that participates in CLUSTER, REPLSET, or GRID. The clients will be allocated across the list of servers.
A macro identifying a list of servers	

- MODE: PROXY or REDIRECT mode. This is an optional parameter.

  - POLICY: Outing policy based on data latency, failure, or workload. This is an optional parameter.
  - WORKERS: The number of worker threads for the SLA. This is an optional parameter.
  - NETTYPE: ETTYPET, HOST, and HOST are for the `sqlhost` entries for the SLA in case the administrator does not want to change the `INFORMIXSQLHOSTS` file. This parameter is optional. “onsoctcp”, “drsoctcp”, and “onsocssl” are the possible values for this entry. “onsoctcp” is the default value.
  - HOST: The host name or P address for the machine. The default is the Connection Manager host name.
  - SERVICE: Service name or port number (no default value). If NETTYPE is specified, SERVICE has to be specified.

– FOC: Fail over controller. This parameter denotes the order in which the Connection Manager should facilitate the failover when the primary goes down. Its syntax is:

```
FOC ORDER=[<server>,...]|DISABLED|NOTIFY TIMEOUT=<n> RETRY=<n>
```

Where:

  - ORDER is a comma separated list of aliases or server types. The failover order will be from the beginning to the end of that list. ORDER can be DISABLED or NOTIFY; if FOC is disabled, CM will ignore the failover request. If FOC is NOTIFY, then CM will send an alarm message through CMALARMPROGRAM.
  - TIMEOUT is the number of seconds to wait before initiating the failover process.
  - RETRY is the number of times to go through the ORDER list until a successful failover is completed. A value of 0 indicates that retries will occur indefinitely. After the RETRY limit has been reached and there is no successful failover, then an alarm will be generated and the failover arbitrator will terminate automated failover.

The default FOC is as follows:

```
FOC  ORDER=SDS, HDR, RSS TIMEOUT=0  RETRY=1
```

  – CMALARMPROGRAM: This parameter points to a shell script that can be used to raise alarms. This is an optional parameter, and can be set at either the global level or for a particular section. If it is defined for a particular section, the scope is for that section. Other section errors or issues will not raise alarms. If FOC is set to NOTIFY, when a failure occurs in one of the nodes and the Connection Manager is monitoring, the Connection Manager will invoke the CMALALRMPROGRAM to raise the alarm.

- In addition, the Connection Manager allows macros to be used for easy substitution instead of having to repeat the content. Example 11-2 shows a macro used in the `ONCMSM` configuration file.

Example 11-2 Macro used in ONCMSM configuration file

```
NAME cm1
MACRO NY=(ny1,ny2,ny3)
MACRO CA=(ca1,ca2,ca3)
LOG 1
LOGFILE {INFORMIXDIR}/tmp/cm1.log
GRID grid1 # grid name
{
INFORMIXSERVER node1,node2,node3
SLA grid1_any DBSERVERS=ANY POLICY=LATENCY
SLA grid1_avail DBSERVERS=${NY},${CA}
}
```

Because client distribution to achieve load balancing is a major task of the Connection Manager, the correct setting of these parameters is essential. It is also important to have correct assignment between the incoming client and the database server, which can serve your needs with respect to the client's range of responsibilities. The configuration file and the associated Informix `sqlhosts` file for the Connection Manager must be set up properly, otherwise, there will be errors when initializing the Connection Manager.

Example 11-3 shows a sample configuration file for a Connection Manager managing a cluster called "mycluster". The INFORMIXSERVER is the server the Connection Manager connects to get information about the "primary" and the "sds" servers to route connections coming in through the sla_prim and sla_sec SLAs.

Example 11-3 Configuration file

```
NAME            cmclust
LOG             1
LOGFILE         /tmp/cm1.log

DEBUG   1
```

```
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_prim   DBSERVERS=primary
        SLA     sla_sec   DBSERVERS=sds
        FOC     ORDER=SDS TIMEOUT=5 RETRY=1
}
```

Example 11-4 shows the required `sqlhosts` entries for the Connection Manager. This `sqlhosts` file has entries for ol_informix1170, sla_prim, and sla_sec. These entries are identified as the list of interesting entities in the configuration file.

Example 11-4 sqlhosts entries for Connection Manager

```
#primary server SQLHOSTS information.
ol_informix1170 onsoctcp informixva ol_informix1170
#SLAs sqlhost information. This is what the clients will connect to.
sla_prim onsoctcp informixva sla_prim
sla_sec  onsoctcp informixva sla_sec
```

Command-line options

You can run the Connection Manager using the ONCMSM command from the `$INFORMIXDIR/bin` directory. Here are the command-line options for ONCMSM:

- -c *<config file name>.*
- -k *<ucm name>*: Deregister and end the Connection Manager.
- -r *<ucm name>*: Refresh the configuration manager.
- -i: This is a Windows only option that registers the new Connection Manager as a service.
- -u: This is a Windows only options that deregisters a running Connection Manager service.

The name of the Connection Manager should be present in the configuration file in the NAME parameter. If not, you will get an error when starting ONCMSM.

For a better illustration of how the command-line options can be combined and which settings they are expecting, some common examples for starting the ONCMSM executable are shown in Example 11-5.

Example 11-5 Command-line examples for ONCMSM

```
#To start ONCMSM
$oncmsm -c ./ucm.config
#To deregister ONCMSM
$oncmsm -k cm1
```

```
#To reconfigure oncmsm
oncmsm -r cm1
```

11.1.3 Configuring Connection Manager on Windows

You are also able to configure and execute the connection manager on the Windows platforms. The ONCMSM binary on Windows is set up a little differently compared to its UNIX equivalents.

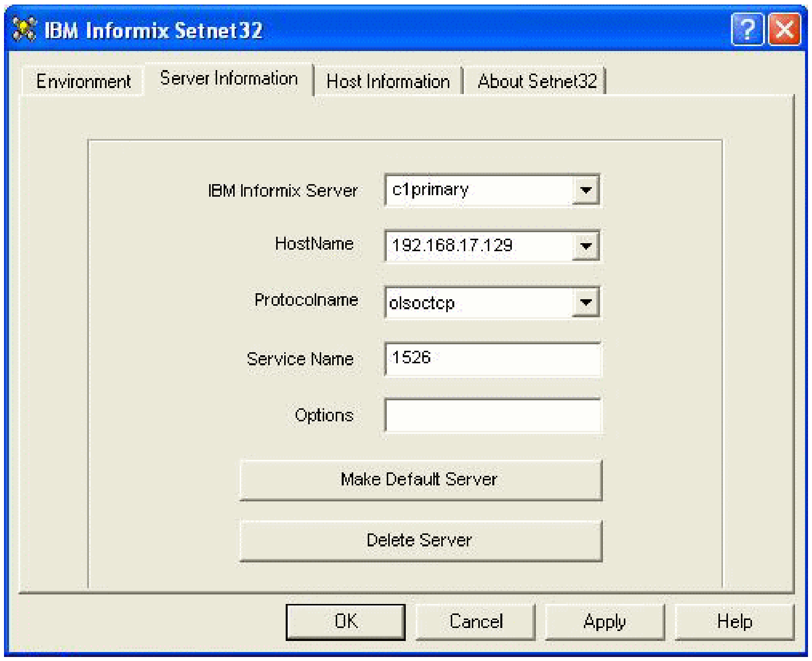
Using setnet for setting up the Connection Manager

As described above, for the Connection Manager to work, the `sqlhost` file must have at least the entries for the primary server and the SLAs on which the clients need to work. The `sqlhost` entries on the Windows platform are set up using the setnet32 utility, which is part of your Client-SDK installation. Example 11-6 shows a cluster definition of an cluster that is monitored by the Connection Manager.

Example 11-6 Sample cluster definition

```
onstat -g cluster

IBM Informix Dynamic Server Version 11.70.UC1DE -- On-Line (Prim) -- Up
02:24:38 -- 177036 Kbytes

Primary Server:c1primary
Current Log Page:6,40
Index page logging status: Enabled
Index page logging was enabled at: 2011/03/07 14:19:27


Server ACKed Log    Supports     Status
       (log, page)  Updates
c1sds1 6,40         Yes          SYNC(SDS),Connected,Active
c1sds2 6,40         Yes          SYNC(SDS),Connected,Active
c1hdr1 6,40         Yes          SYNC(HDR),Connected,On
c1rss1 6,40         Yes          ASYNC(RSS),Connected,Active
c1rss2 6,40         Yes          ASYNC(RSS),Connected,Active
```

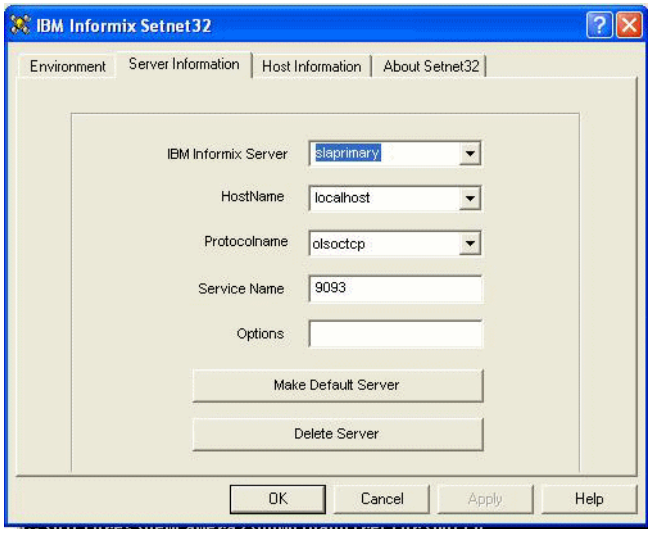
For the example cluster, we want to set a setnet32 entry for the Connection Manager to connect to the primary (c1primary) and define two SLAs to which the applications can connect, one for connecting to the primary and the second for connecting to the SDS secondaries (c1sds1 and c1sds2). Each of these entries are set using the setnet32 utility. Figure 11-2 shows the connection entry for the primary server.



IBM Informix Server: c1primary
HostName: 192.168.17.129
Protocolname: olsoctcp
Service Name: 1526
Options:

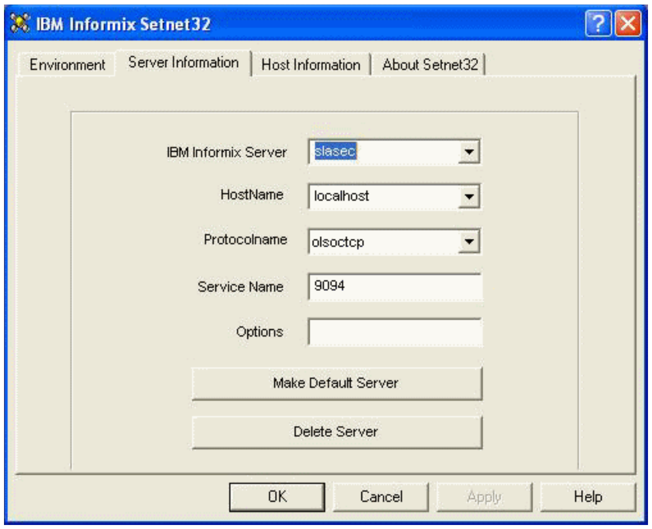
Figure 11-2 Primary connection

Figure 11-3 shows the SLA for connecting to the primary server in the cluster and Figure 11-4 shows the SLA for connecting to the secondary servers.



IBM Informix Server: slaprimary
HostName: localhost
Protocolname: olsoctcp
Service Name: 9093
Options:

Figure 11-3 SLA for connecting to the primary server in the cluster



IBM Informix Server: slasec
HostName: localhost
Protocolname: olsoctcp
Service Name: 9094
Options:

Figure 11-4 SLA for connecting to the secondary servers

After the setnet32 configuration is done, the next step is to define a configuration file for the Connection Manager.

Configuration and maintenance of the ONCMSM service

The binary oncmsm.exe can be run using a configuration file passed in with the -c parameter. If a configuration file is not specified, a default configuration file, `cmsm.cfg`, located in `%INFORMIXDIR%\etc\`, is used. If the default file is not present and no configuration file is specified at launch, an error is thrown when executing the binary, as shown in Example 11-7.

Example 11-7 Error when default configuration file is absent

```
C:\Program Files\IBM\Informix\11.70\bin>oncmsm
13:56:03 Invalid configuration file path
"C:\PROGRA~1\IBM\Informix\11.70\etc\cmsm.cfg"
```

After the proper configuration file is given as input and the Connection Manager starts, the output messages are logged into the log file specified in the configuration file. Example 11-8 shows a sample configuration file and output from the Connection Manager log files. In this case, the `cmsm.cfg` file exists at the default location.

Example 11-8 Sample windows output

```
C:\Program Files\IBM\Informix\11.70\etc>type cmsm.cfg
NAME wincm
LOG 1
LOGFILE c:\temp\wincm.log
DEBUG 1

CLUSTER c1
{
        INFORMIXSERVER  c1primary
        SLA     slaprimary   DBSERVERS=primary
        SLA     slasec   DBSERVERS=sds
}


C:\Program Files\IBM\Informix\11.70\bin>oncmsm
14:00:07 IBM Informix Connection Manager log file: c:\temp\wincm.log

C:\Program Files\IBM\Informix\11.70\bin>tail c:\temp\wincm.log
14:10:37 Connection Manager successfully connected to c1primary
14:10:37 cmsm_load_sysha adding c1hdr1 localhost 1528 [w:/tristarm/./asf/cmsm/cm
sm_server.ec:571]
14:10:37 create new thread 9112 for c1hdr1 [W:\tristarm\.\asf\cmsm\cmsm_sla.c:72
7]
14:10:37 cmsm_load_sysha adding c1primary 192.168.17.129 1526 [w:/tristarm/./asf
/cmsm/cmsm_server.ec:571]
14:10:37 Cluster c1 Arbitrator FOC string = ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
14:10:37 Cluster c1 Arbitrator setting primary name = c1primary [W:\tristarm\.\a
sf\cmsm\cmsm_arb.c:503]
14:10:37 FOC[0] = SDS
14:10:37 FOC[1] = HDR
14:10:37 FOC[2] = RSS
14:10:37 FOC TIMEOUT = 0
14:10:37 FOC RETRY = 1
...
14:10:38 Connection Manager started successfully

#onstat -g cmsm output
```

```
onstat -g cmsm

IBM Informix Dynamic Server Version 11.70.UC1DE -- On-Line (Prim) -- Up 04:33:19 -- 177036 Kbytes
Connection Manager Name: wincm
        Hostname: ibm-thinkvg.svl.ibm.com

        SLA                    Connections   Service/Protocol   Rule
        slaprimary                       0      9093/olsoctcp   DBSERVERS=primary
        slasec                           0      9094/olsoctcp   DBSERVERS=sds

Failover Configuration:
Connection Manager name        Rule                Timeout   State
wincm
                               ORDER=SDS                 0   Active Arbitrator, Primary is up
```

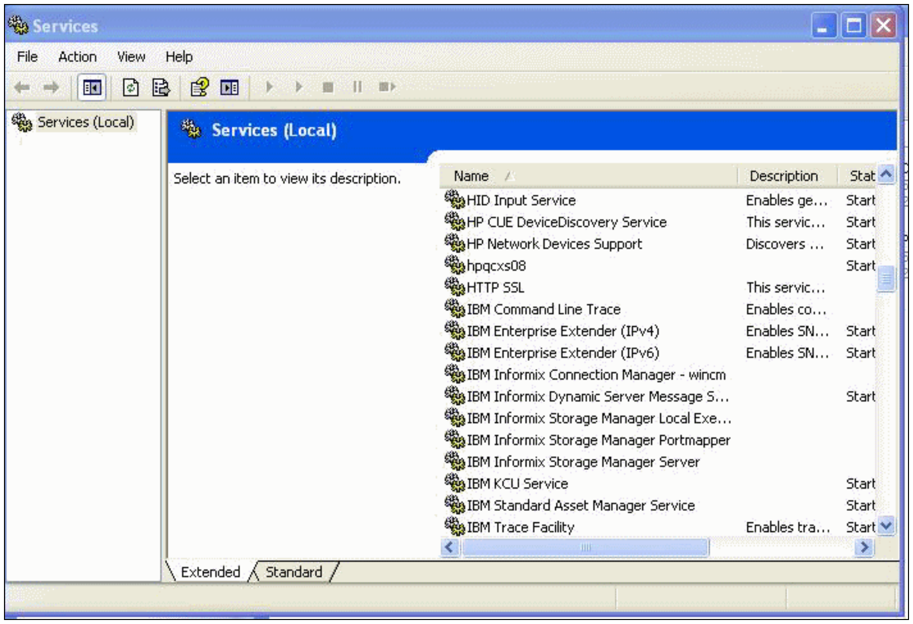
The Connection Manager can also be installed and maintained on Windows as a service. An intermediate step is necessary before you can successfully start the service. You have to use the -i option to create the new service. Keep in mind that only the -i option is accepted for definition purposes; all other parameters have to be specified in the configuration file used by default at creation time. Example 11-9 show the command to register the Connection Manager as a service.

Example 11-9 Registering Connection Manager as a service

```
C:\Program Files\IBM\Informix\11.70\bin>oncmsm -i
14:50:10 IBM Informix Connection Manager log file: c:\temp\wincm.log
Connection Manager Service "wincm" installed successfully
```

After the Connection Manager is registered as a service, you should be able to see it in the list of services and be able to start and stop it from the Service Control Panel in Windows. You can check whether it is registered by selecting **Control Panel** → **Administrative Tools** and opening the **Services** tab.

Figure 11-5 shows that wincm is now a registered service.



Name	Description	Stat
HID Input Service	Enables ge...	Start
HP CUE DeviceDiscovery Service	This servic...	Start
HP Network Devices Support	Discovers ...	Start
hpqcxs08		Start
HTTP SSL	This servic...	
IBM Command Line Trace	Enables co...	
IBM Enterprise Extender (IPv4)	Enables SN...	Start
IBM Enterprise Extender (IPv6)	Enables SN...	Start
IBM Informix Connection Manager - wincm		
IBM Informix Dynamic Server Message S...		Start
IBM Informix Storage Manager Local Exe...		
IBM Informix Storage Manager Portmapper		
IBM Informix Storage Manager Server		
IBM KCU Service		Start
IBM Standard Asset Manager Service		Start
IBM Trace Facility	Enables tra...	Start

Figure 11-5 Connection Manager as a service

After the service has been registered, the ONCMSM base service can be stopped and removed from the list of currently maintained services on the local machine. You can use the -k option provided by the ONCMSM to stop the service. Use the -u option to remove the service from the service list.

11.1.4 Client application requirements

In this section, we discuss certain requirements that are related to the client application development phase and application build environment that must be met to successfully use the Connection Manager's capabilities.

Client SDK Version 3.50

To benefit from Connection Manager, client applications have to be compiled at least with Client SDK Version 3.50. All clients that are not based on this Client SDK version cannot connect to the ONCMSM utility. Any attempt to connect will return a -908 error, even if the client uses a properly configured `sqlhosts` file.

The behavior for a simple client attempting to connect to an SLA defined by Connection Manager is shown in Example 11-10.

Example 11-10 Different behavior depending on the clients SQL version

```
#Connection Manager was started with
$oncmsm -c ./ucm.config
#client application environment
INFORMIXDIR=/opt/IBM/informix
INFORMIXSERVER=sla_prim
INFORMIXSQLHOSTS=/home/informix/setupdir/sqlhosts.client

#client code -- once compiled with CSDK 3.50 and with 2.90
$include "sqlca.h";
$include "sqlhdr.h";
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


main(int argc,char *argv[])
{
$begin declare section;
char buffer[100];
int customer_num;
char fname[20],lname[20];
$end declare section;


sprintf(buffer,"stores_demo@%s",argv[1]);

$connect to :buffer;
printf("SQLCODE after connect %d\n",sqlca.sqlcode);
printf("GETCHAR - please press a character to continue\n");
getchar();
$prepare cstmt from "select customer_num,fname,lname from customer";
$declare c1 cursor for cstmt;
$open c1;
while(sqlca.sqlcode ==0)
{
        $fetch c1 into :customer_num,:fname,:lname;
        printf("CUST DETAILS %d %s %s\n",customer_num,fname,lname);
}
$close c1;
$disconnect all;
}

#sqlhosts
ol_informix1170 onsoctcp informixva ol_informix1170
```

```
sla_prim onsoctcp informixva sla_prim
sla_sec  onsoctcp informixva sla_sec


$> main_350 sla_prim
Connect: 0

$> main_290 sla_prim
Connect: -908
```

The client has to connect to an SLA name

The Connection Manager defines the SLA as the interface for the client application. The SLA is represented to the client by a name. In ONCMSM, the name is finally substituted on the target database server for the correct one, based on the given directions at start time. To successfully use the Connection Manager, the client application has to use the SLA name in the connect statement. Coming back to the simple esql/c program described in Example 11-10 on page 427, you can see the expected behavior. The application executes the connection with the first program option. You have to specify either sla_prim or sla_sec, as these are the SLAs defined in the `INFORMIXSQLHOSTS` file. ONCMSM will then direct the client to the appropriate back-end server.

Redirecting the client in a server outage

At connection time of a client application, with the help of a Connection Manager, the application is assigned to a certain database server in the cluster environment. The assignment is based on the defined SLA where the client is connected. If there is an outage of the current database server, the client receives a connection error either for the currently executed SQL statement or the next statement that attempts to execute. All open cursors and prepared statements become invalid. But for the redirection, the client does not have to be restarted and no environment variables have to be changed. Only a new connect statement, as the result of an error handling process in the application, is required. The Connection Manager redirects the application to another database server if one is available. Example 11-11 shows an example of a simple esql/c client handling a reconnect.

Example 11-11 Using client redirection in case of a database server outage

```
#environment
INFORMIXDIR=/opt/IBM/informix
INFORMIXSERVER=sla_prim
INFORMIXSQLHOSTS=/home/informix/setupdir/sqlhosts.client

$include sqlca;
#include <stdio.h>
```

```
main(int argc, char**argv)
$char buffer[100];
$int customer=0;
sprintf(buffer,"stores_demo@%s",argv[1]);
$connect to :buffer;
printf("Connect:%ld \n",sqlca.sqlcode);
$declare c1 cursor for select customer_num from customer ;
$open c1;
$fetch c1 into :customer;
while ( sqlca.sqlcode ==0L) {
$fetch c1 into :customer;
printf("Customer : %d \n",customer);
};

/* assume a connection problem due a server outage */
if ( sqlca.sqlcode != 100 ) {
$connect to :buffer;
printf("Connect:%ld \n",sqlca.sqlcode);
   $declare c1 cursor for select customer_num from customer and
      customer_num>:customer;
   $open c1;
   $fetch c1 into :customer;
   while ( sqlca.sqlcode ==0L) {
      printf("Customer : %d \n",customer);
      $fetch c1 into :customer;
   }
}
$disconnect all;
}
```

11.1.5 Security considerations at the start time of the ONCMSM

User accounts running client applications against the database server can be authenticated in two ways on the server: Either they are working in a trusted environment or they are authenticated in the application by providing a user and a password at connection time. Trusted environments are all local accounts, and have remote connections that use the `.rhosts` or `hosts.equiv` trusting mechanisms provided by the operating system.

The Connection Manager can also be considered a special client application. Because it can be installed either on the same machine as the target database server on an intermediate machine between the client and the database server or at the client application machine, the requirements are the same for trusted and non-trusted behavior. Running the Connection Manager in a non-trusted environment requires that you provide the executable in a certain way; the information about the password and the user is on one side and the database server name is the target on the other side. This has be done using an encrypted file for security reasons.

You have to execute the following steps to create the file:

- Create a plain text file with the user, password, and connection details.
- Encrypt the file using the onpassword utility.
- Start the Connection Manager, which should successfully start without any -951 errors in the log file.

In the following sections, we give more details about the content of the password file, how to encrypt the password file, and how you can monitor it to see that your changes are taken into account after restarting ONCMSM.

$INFORMIXDIR/etc/passwd_file

To establish a password and user management for a non-trusted start of the Connection Manager, you must create a plain text file with the connection and password details of the users in the first step. The file only needs to contain the details for the particular users that have to start the Connection Manager. Because you will be working with sensitive user specific data, be sure that the place where you have created the file has secure file and directory permissions.

Example 11-12 shows a sample `passwd_file`. In the sample environment, we define two separate Connection Managers in the same environment. They maintain different cluster environments using the name of the primaries, c_prim and c_alternate_prim. The Connection Manager has to be started with different environments and configuration files, as shown in the example. The Connection Manager must be started by different users, that is, cm1_user and cm2_user. Because we start the ONCMSM from the same installation, we have to add the user for both ONCMSM instances to the same `passwd_file`.

Example 11-12 Example of a passwd_file and the relationship to the ONCMSM config

```
#Environment for connection manager 1
export INFORMIXDIR=/opt/informix
export INFORMIXSERVER=c_prim

#ONCMSM configuration file for connection manager 1
#we want to startup the ONCMSM with cm1_user not trusted
```

```
NAME cm1
SLA oltp1=Primary
LOGFILE /tmp/logfile_cm1

#Environment for connection manager 2
export INFORMIXDIR=/opt/informix
export INFORMIXSERVER=c_alternate_prim

#ONCMSM configuration file for connection manager 2
#we want to startup the ONCMSM with cm2_user not trusted
NAME cm2
SLA oltp2=Primary
LOGFILE /tmp/logfile_cm2

#sqlhosts file
c_prim onsoctcp host 20000
c_alternate_prim onsoctcp host1 20001
oltp1 onsoctcp localhost 9000
oltp2 onsoctcp localhost 9001

# of the plain password text file
c_prim c_alternate_prim cm1_user password1
c_alternate_prim c_prim cm2_user password99
```

The onpassword utility

ONCMSM uses an encrypted file, `$INFORMIXDIR/etc/passwd_file`, to contain the users and their passwords in a non-trusted environment. To encrypt the password in the plain text file created in the previous section, you have to use the onpassword utility. The onpassword utility is part of the Client SDK installation, which, as mentioned before, comes with the server bundle or as a stand-alone installation package.

To successfully execute the tool, you must use the following options:

- -k <accesskey>, to provide a key for the encryption and decryption of the file.
- -e, which is used with the name of the text file with the users, if you want to create the encrypted target file named `$INFORMIXDIR/etc/passwd_file`.
- -d, which is used with the name of an output file, if you want to create a plain text file from the content of the currently existing `$INFORMIXDIR/etc/passwd_file`.

Example 11-13 shows some examples of how to use onpassword in both directions.

Example 11-13 How to use onpassword for encrypting and decrypting for password file

```
$cat passwdfile
serv1 serv2 informix password1
serv2 serv1 informix password1

#encrypt a plain text file
$onpassword -k 123456 -e passwdfile
$ls -al $INFORMIXDIR/etc/passwd
-rw-r--r--   1 informix informix 113 Feb 21 19:47/11.50/etc/passwd_file

#decrypt the existing $INFORMIXDIR/etc/passwd to recover a lost text
$onpassword -k 123456 -d passwdfile.decrypt
```

Restarting the ONCMSM

After creating the encrypted passwd_file, you should be able to successfully start the Connection Manager in a non-trusted environment. To do so, complete the following steps:

1. Execute **`oncmsm -c <your config file>`**.
2. Verify that the password management is in place.

For verification that the changes were accomplished, look in the ONCMSM log file. Example 11-14 shows two example outputs. The first output shows the incorrect password and the second one shows the correct password.

Example 11-14 onpassword used with the incorrect/correct password

```
#UCM Configuration file (ucm.config)
NAME            cm1
LOG             1
LOGFILE         /tmp/cm1.log

DEBUG   1
GRID mygrid
{
        INFORMIXSERVER  serv1
        SLA     sla_any   DBSERVERS=serv1,serv2,serv3
}

#Onpassword file contains encrypted password to connect to serv1 and serv2

#oncmsm -c ./ucm.config

#oncmsm run with incorrect password note -951 errors
```

```
13:41:59 Connection Manager successfully connected to serv3
13:41:59 cmsm_load_sysha adding serv3 hal14.lenexa.ibm.com serv3 [cmsm_server.ec:555]
13:41:59 server serv3 version is 11.70 [cmsm_er.ec:2137]
13:41:59 set CM version failed, sqlcode = (-937,0,cdrcmd) [cmsm_server.ec:1366]
13:41:59 CONNECT to @serv2|onsoctcp|hal14.lenexa.ibm.com|serv2 AS serv20 USER informix
SQLCODE = (-951,0,informix@ibm-thinkvg.lenexa.ibm.com[informixva]) [cmsm_server.ec:155]
13:41:59 Connection Manager started successfully
13:42:00 CONNECT to @serv2|onsoctcp|hal14.lenexa.ibm.com|serv2 AS serv20 USER informix
SQLCODE = (-951,0,informix@ibm-thinkvg.lenexa.ibm.com[informixva]) [cmsm_server.ec:155]
13:42:00 CONNECT to @serv2|onsoctcp|hal14.lenexa.ibm.com|serv2 AS serv20 USER informix
SQLCODE = (-951,0,informix@ibm-thinkvg.lenexa.ibm.com[informixva]) [cmsm_server.ec:155]


#oncmsm run with correct password
#oncmsm -c ./ucm.config

13:42:37 CONNECT to @serv2|onsoctcp|hal14.lenexa.ibm.com|serv2 AS serv20 USER informix
SQLCODE = (0,0,) [cmsm_server.ec:155]
13:42:37 database sysmaster @serv2|onsoctcp|hal14.lenexa.ibm.com|serv2 AS serv20 USER
informix SQLCODE = (0,0,) [cmsm_server.ec:164]
13:42:37 serv2 protocols = 1 [cmsm_server.ec:1741]
13:42:37 SQL fetch sysconfig_cursor sqlcode = (100,0,) [cmsm_server.ec:1788]
13:42:37 Connection Manager successfully connected to serv2
```

If the `$INFORMIXDIR/etc/passwd_file` exists but is not correct, for example, it contains the plain text instead of the encrypted text, then the ONCMSM attempts to access the database server again in trusted mode. The log file contains the `Warning: Password Manager failed, working in trusted node` message, but no reason is given.

11.1.6 Monitoring the ONCMSM status

Both the Informix database server and the ONCMSM executable provide the DBA with monitoring functionality. Looking for additional information about the Connection Manager in the database server requires that the ONCMSM be started successfully against the database cluster or grid.

Monitoring the Connection Manager in Informix

The database server provides three different sources for getting more information about the Connection Manager status. You may be able to find additional information using the onstat utility, the sysmaster interface, or by looking into the database server's `online.log` file. In the following sections, we show the common outputs and messages used by the ONCMSM interface in the database server.

The onstat interface

To obtain more information about the database server currently registered as the Connection Manager, run **`onstat -g cmsm`**. Example 11-15 shows a sample output of Informix with the single connection manager used in the test environment.

Example 11-15 onstat -g cmsm

```
#oncmsm configuration file - Note we are now working with a grid and cluster
using a single Connection Manager.
NAME           cm1
LOG            1
LOGFILE        /tmp/cm1.log

DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_prim   DBSERVERS=primary
        SLA     sla_sec   DBSERVERS=sds
        FOC     ORDER=SDS TIMEOUT=5 RETRY=1
}
GRID mygrid
{
        INFORMIXSERVER serv1,serv2,serv3
        SLA     sla_any   DBSERVERS=serv1,serv2,serv3 POLICY=LATENCY
}


$onstat -g cmsm
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 22:59:45 -- 160540 Kbytes
Connection Manager Name: cm1
       Hostname: informixva

       SLA                    Connections   Service/Protocol   Rule
       sla_prim/CLUSTER/mycluster       0   sla_prim/onsoctcp  DBSERVERS=primary
       sla_sec/CLUSTER/mycluster        0   sla_sec/onsoctcp   DBSERVERS=sds
       sla_any/SERVERSET/mygrid         0   sla_any/onsoctcp DBSERVERS=serv1,serv2,serv3

Failover Configuration:
Connection Manager name        Rule              Timeout   State
cm1                            ORDER=SDS               0
```

The sysmaster interface in the Informix server

You can obtain similar information from the syscmsmtab, syscmsmsla, and syscmsm tables. Example 11-16 shows a sample output.

Example 11-16 Using the sysmaster interface for monitoring Connection Managers

```
$dbaccess -e sysmaster << EOF
select * from syscmsmtab;
select * from syscmsm;
select * from syscmsmsla;
EOF

Database selected.


address  1263754960
sid      111
name     cm1
host     informixva
foc      ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
flag     256


1 row(s) retrieved.
sid         111
name        cm1
host        informixva
foc         ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
flag        256
sla_id      0
sla_name    sla_prim/CLUSTER/mycluster
sla_define  DBSERVERS=primary
connections 0

sid         111
name        cm1
host        informixva
foc         ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
flag        256
sla_id      1
sla_name    sla_sec/CLUSTER/mycluster
sla_define  DBSERVERS=sds
connections 0

sid         111
name        cm1
host        informixva
foc         ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
flag        256
sla_id      2
sla_name    sla_any/SERVERSET/mygrid
sla_define  DBSERVERS=serv1,serv2,serv3
connections 0

3 row(s) retrieved.

address     1263738112
sid         111
sla_id      0
sla_name    sla_prim/CLUSTER/mycluster
sla_define  DBSERVERS=primary
connections 0

address     1264276832
sid         111
sla_id      1
sla_name    sla_sec/CLUSTER/mycluster
sla_define  DBSERVERS=sds
connections 0

address     1264277200
sid         111
sla_id      2
sla_name    sla_any/SERVERSET/mygrid
```

```
sla_define   DBSERVERS=serv1,serv2,serv3
connections  0

3 row(s) retrieved.
```

The online.log file

Basic information about the connected Connection Managers for a specific database server in a cluster environment can also be obtained from the `online.log` file. Common messages referring to the Connection Manager appear as shown in Example 11-17.

Example 11-17 Possible oncmsm related messages in the Online.log

```
20:16:43  CM:Connection manager cm1 de-registered with the server
21:12:59  CM:Connection manager cm1 terminated abnormally
21:14:07  CM:Connection manager cm1 registered with the server
```

Monitoring the Connection Manager in the ONCMSM log file

When the DEBUG flag is set to 1 in the configuration file, ONCMSM outputs additional information. This information, along with the regular trace information, can be used to understand the state of ONCMSM. Example 11-18 shows an example of the log file after a successful startup of the ONCMSM. The lines with the source references are not written to the log file when the DEBUG parameter is not specified or set to 0 in the configuration file.

Example 11-18 Example output from oncmsm log file taken after a startup of the oncmsm

```
15:01:38 IBM Informix Connection Manager
15:01:38 add type primary to SLA sla_prim [cmsm_sla.c:1820]
15:01:38 SLA sla_prim listener mode REDIRECT
15:01:38 add type sds to SLA sla_sec [cmsm_sla.c:1820]
15:01:38 SLA sla_sec listener mode REDIRECT
15:01:38 SLA sla_any type 0x8356308 add 0x8356328 [cmsm_sla.c:1885]
15:01:38 add server serv1 to SLA sla_any [cmsm_sla.c:1887]
15:01:38 SLA sla_any type 0x8356308 add 0x8356348 [cmsm_sla.c:1885]
15:01:38 add server serv2 to SLA sla_any [cmsm_sla.c:1887]
15:01:38 SLA sla_any type 0x8356308 add 0x8356368 [cmsm_sla.c:1885]
15:01:38 add server serv3 to SLA sla_any [cmsm_sla.c:1887]
15:01:38 SLA sla_any listener mode REDIRECT
15:01:38 Connection Manager name is cm1
15:01:38 Starting Connection Manager...
15:01:38 set the maximum number of file descriptors 32767 failed
15:01:38 setrlimit(RLIMIT_NOFILE, 32767) = -1, errno = 0 [cmsm_main.c:3723]
15:01:38 the current maximum number of file descriptors is 4096
15:01:38 fcntl(/opt/IBM/informix/tmp/cmsm.pid.cm1) success errno = 0 [cmsm_main.c:1969]
15:01:38 dbservername = ol_informix1170
15:01:38 nettype      = onsoctcp
15:01:38 hostname     = informixva
15:01:38 servicename  = ol_informix1170
15:01:38 options      =
15:01:38 switch to daemon mode
15:01:38 new daemon pid is 19152 [cmsm_main.c:3638]
15:01:38 create new thread -1231135856 for ol_informix1170 [cmsm_sla.c:728]
15:01:38 listener sla_prim initializing
15:01:38 listener sla_sec initializing
15:01:38 Listener sla_sec DBSERVERS=sds is active with 8 worker threads
```

```
15:01:38 Listener sla_prim DBSERVERS=primary is active with 8 worker threads
15:01:38 CONNECT to @ol_informix1170|onsoctcp|informixva|ol_informix1170 AS ol_informix11700
SQLCODE = (0,0,) [cmsm_server.ec:155]
15:01:38 database sysmaster @ol_informix1170|onsoctcp|informixva|ol_informix1170 AS ol_informix11700
SQLCODE = (0,0,) [cmsm_server.ec:164]
15:01:38 ol_informix1170 protocols = 1 [cmsm_server.ec:1741]
15:01:38 adding ol_informix1170 alias dr_informix1170 with protocol drsoctcp 3 [cmsm_server.ec:1841]
15:01:39 SQL fetch sysconfig_cursor sqlcode = (100,0,) [cmsm_server.ec:1788]
15:01:39 Connection Manager successfully connected to ol_informix1170
```

11.1.7 Example infrastructures maintained by a Connection Manager

Now that you have an overview about how to set up and monitor the ONCMSM, we now describe, based on sample Informix cluster infrastructures, how the connection management works to provide you a better understanding of how the components work together.

Pure SDS infrastructure with one primary and two secondary

We start this discussion with a simple Informix cluster environment. Imagine that you want to use the SDS infrastructure to lower the cost of disk space and share the dbspace layout over several database servers. We can enhance these benefits by adding an automatic client balancing across the SDS nodes under the control of the Connection Manager. Figure 11-6 shows this simple cluster environment.

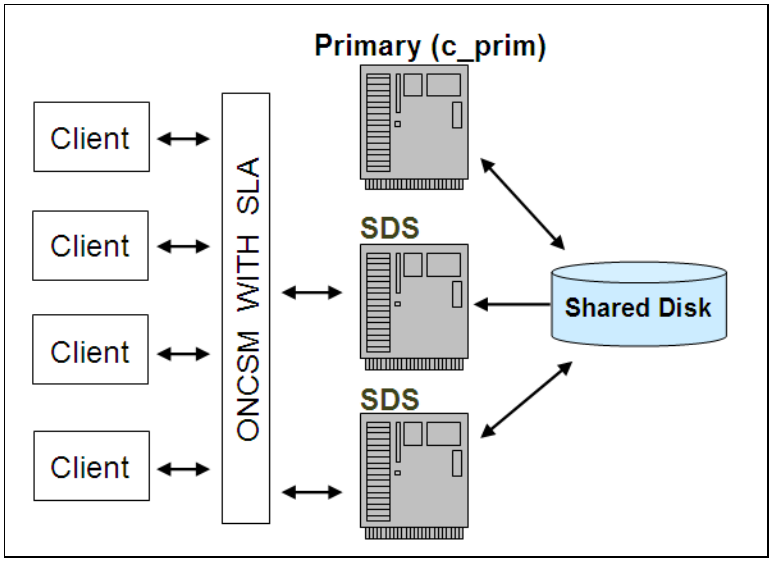


Figure 11-6 Simple SDS cluster with a primary and two SDS servers

To embed the Connection Manager successfully between the existing cluster environment and the database clients, complete the following steps:

1. Add the SLA entries to the appropriate `sqlhosts` file.
2. Set the environment variables for the Connection Manager.
3. Start the Connection Manager with the defined directions for the SLA.
4. Check that the Connection Manager started successfully on the database server.
5. Start the client. The connection should be done according to the SLA instead of using the database server name specified in the `sqlhosts` file.
6. Check the distribution of the clients.

Example 11-19 shows the steps of the configuration for the ONCMSM and illustrates what happens when the clients are connecting. In the first step, after setting up the environment, the Connection Manager is started with an SLA definition to spread the incoming client across the SDS secondaries. You can monitor the successful start by running **`onstat -g cmsm`**. Some simple client connections to the stores_demo database are started. The clients connect to the SLA named "sla_sec". You can see that they are equally distributed. Observe each output.

Notice that for the FOC (failover configuration) parameter, the default setting was applied because we have not specified anything. For directions about how to configure this particular parameter and the meaning of the settings, refer to 11.2.1, "Failover configuration" on page 449.

Example 11-19 Client balancing on a simple SDS cluster environment

```
#create the sqlhosts file
ol_informix1170 onsoctcp informixva ol_informix1170
sla_prim onsoctcp informixva sla_prim
sla_sec  onsoctcp informixva sla_sec

#set the required environment
INFORMIXDIR=/opt/IBM/informix
INFORMIXSERVER=sla_prim
INFORMIXSQLHOSTS=/home/informix/setupdir/sqlhosts.cm
export PATH=$PATH:$INFORMIXDIR/bin

#start oncmsm
$oncmsm -c ./ucm.config

#Verify the setting and the successful startup
$> onstat -g cmsm
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 1 days 07:30:24 -- 160540
Kbytes
```

```
Connection Manager Name: cm1
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_prim/CLUSTER/mycluster      0    sla_prim/onsoctcp   DBSERVERS=primary
        sla_sec/CLUSTER/mycluster       4   sla_sec/onsoctcp   DBSERVERS=sds

Failover Configuration:
Connection Manager name        Rule               Timeout   State
cm1                           ORDER=SDS                0


#Run some clients using the program from Example 11-11 on page 428
# For this example run the application from 4 different terminal windows
#Terminal 1 - testapp sla_sec
#Terminal 2 - testapp sla_sec
#Terminal 3 - testapp sla_sec
#Terminal 4 - testapp sla_sec

#Monitoring the SDS secondaries
# watch out for the stores_demo based database sessions
#onstat -g sql on first secondary
$onstat -g sql
IBM Informix Dynamic Server Version 11.70.UC2 -- Read-Only (SDS) -- Up 1 days 07:23:10
-- 152348 Kbytes


Sess       SQL            Current           Iso Lock       SQL  ISAM F.E.
Id         Stmt type      Database          Lvl Mode       ERR  ERR  Vers  Explain
73         -              stores_demo       DR  Not Wait   0    0    9.37  Off
72         -              stores_demo       DR  Not Wait   0    0    9.37  Off
71         SELECT         sysmaster         DR  Not Wait   0    0    9.37  Off
4          -              -                 -   Not Wait   0    0    9.24  Off

#Session information for the second secondary
IBM Informix Dynamic Server Version 11.70.UC2 -- Read-Only (SDS) -- Up 1 days 07:23:09
-- 152348 Kbytes


Sess       SQL            Current           Iso Lock       SQL  ISAM F.E.
Id         Stmt type      Database          Lvl Mode       ERR  ERR  Vers  Explain
65         -              stores_demo       DR  Not Wait   0    0    9.37  Off
64         -              stores_demo       DR  Not Wait   0    0    9.37  Off
63         SELECT         sysmaster         DR  Not Wait   0    0    9.37  Off
6          -              -                 -   Not Wait   0    0    9.24  Off
```

Defining multiple entities in one Connection Manager

In contrast to the previous infrastructure, where we defined only one cluster, we now extend the same Connection Manager to manage a cluster and a grid. The cluster SLAs are the same as before, and the grid SLA directs clients to any node in the grid. Figure 11-7 shows the layout.

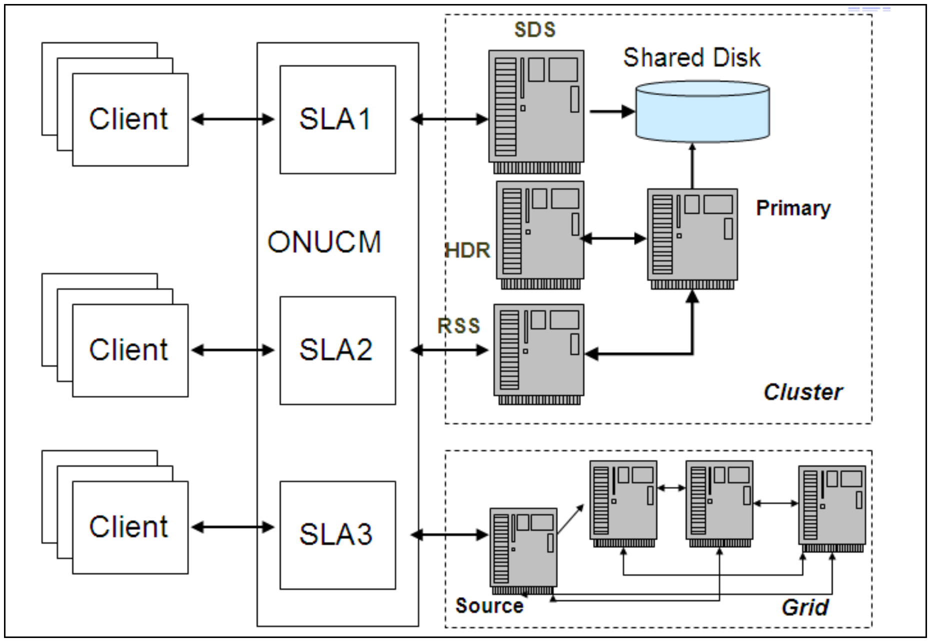


Figure 11-7 Informix cluster environment with multiple SLAs defined in one ONCMSM

All of the grid and cluster information is maintained in a single ONCMSM configuration file, as shown in Example 11-15 on page 434. After a successful start of the Connection Manager, you can see multiple lines in the **onstat -g cmsm** output. All of them refer to the same Connection Manager instance. You can see the SLA for each one. The clients are directed to the appropriate node both in the grid and cluster by the Connection Manager. Example 11-20 shows the output of **onstat -g cmsm** and the client distribution.

Example 11-20 One oncmsm and multiple SLA definitions

```
#sqlhosts

ol_informix1170 onsoctcp informixva ol_informix1170
sla_prim onsoctcp informixva sla_prim
sla_sec  onsoctcp informixva sla_sec
sla_any onsoctcp informixva sla_any
utm_group_1 group - - i=1
serv1 onsoctcp hal14.lenexa.ibm.com serv1 g=utm_group_1
```

```
utm_group_2 group - - i=2
serv2 onsoctcp hal14.lenexa.ibm.com serv2 g=utm_group_2
utm_group_3 group - - i=3
serv3 onsoctcp hal14.lenexa.ibm.com serv3 g=utm_group_3

#environment
export INFORMIXDIR=/opt/informix
export INFORMIXSERVER=sla_prim

#ONCMSM startup
#The Connection Manager configuration file has information for a cluster and a
grid as in Example 11-15 on page 434.
$ONCMSM -c ucm.config

informix@informixva[ol_informix1170]:~/setupdir$ onstat -g cmsm

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 1 days 20:36:16 -- 160540 Kbytes
Connection Manager Name: cm1
        Hostname: informixva

        SLA                    Connections   Service/Protocol    Rule
        sla_prim/CLUSTER/mycluster       0   sla_prim/onsoctcp   DBSERVERS=primary
        sla_sec/CLUSTER/mycluster        0   sla_sec/onsoctcp    DBSERVERS=sds
        sla_any/SERVERSET/mygrid         0   sla_any/onsoctcp
                                         DBSERVERS=utm_group_1,utm_group_2,utm_group_3 POLICY=LATENCY

Failover Configuration:
Connection Manager name        Rule              Timeout   State
cm1                            ORDER=SDS               0


#starting the client - as we saw the distribution with the cluster earlier, we
will show only the distribution with the grid here.
testapp sla_any &; testapp sla_any &; testapp sla_any &; testapp sla_any &

#information on serv1
$onstat -g sql
onstat -g sql

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 1 days 19:16:22 -- 168732
Kbytes


Sess       SQL            Current           Iso Lock       SQL  ISAM F.E.
Id         Stmt type      Database          Lvl Mode       ERR  ERR  Vers  Explain
855        SELECT         sysadmin          CR  Not Wait   0    0    9.22  Off
854        -              sysmaster         CR  Not Wait   0    0    9.22  Off
846        -              stores_demo       CR  Not Wait   0    0    9.37  Off
845        -              stores_demo       CR  Not Wait   0    0    9.37  Off
844        -              stores_demo       CR  Not Wait   0    0    9.37  Off
843        SELECT         sysmaster         DR  Not Wait   0    0    9.37  Off
839        SELECT         sysmaster         DR  Not Wait   0    0    9.37  Off
93                        syscdr            CR  Wait 30    0    0    9.03  Off
27                        sysadmin          DR  Wait 5     0    0    -     Off
```

```
26                        sysadmin          DR  Wait 5     0    0    -     Off
25                        sysadmin          DR  Wait 5     0    0    -     Off

#information on serv3
$onstat -g sql
onstat -g sql

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 1 days 19:03:58 -- 160540
Kbytes


Sess       SQL            Current           Iso Lock       SQL  ISAM F.E.
Id         Stmt type      Database          Lvl Mode       ERR  ERR  Vers  Explain
752        SELECT         sysadmin          CR  Not Wait   0    0    9.22  Off
751        -              sysmaster         CR  Not Wait   0    0    9.22  Off
747        -              stores_demo       CR  Not Wait   0    0    9.37  Off
745        SELECT         sysmaster         DR  Not Wait   0    0    9.37  Off
27                        sysadmin          DR  Wait 5     0    0    -     Off
26                        sysadmin          DR  Wait 5     0    0    -     Off
25                        sysadmin          DR  Wait 5     0    0    -     Off
```

Defining a SLA group to ensure maximum availability

Finally, we discuss a completely different view of the Connection Manager configuration. Similar to the combination of database servers into server groups, it is possible to combine SLA definitions into groups. The benefit of using groups at the connection time of the clients is that you can skip all unavailable SLAs in the group and connect to the first SLA that is available.

This configuration certainly only makes sense for SLAs defined in different Connection Managers. If they are defined in the same Connection Manager and this one is not started, the client is not able to connect to any of the SLAs defined in the group. This particular failover strategy is implemented in the Informix connection layer of each database client. It is provided by a library that is automatically linked to the application at build time for statically linked clients, or dynamically loaded at startup time. This configuration prevents the clients from performing manual trial and error on their own.

Figure 11-8 shows the global picture of this particular configuration.

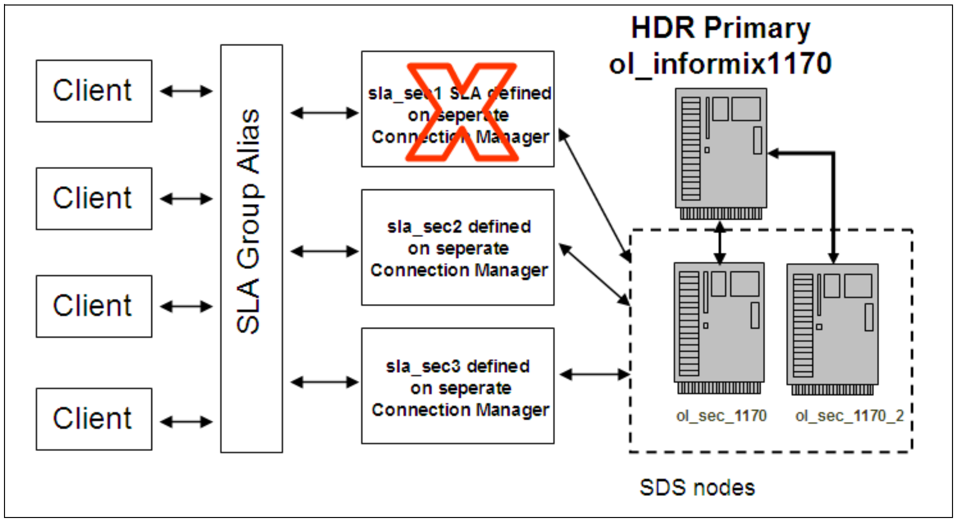

HDR Primary
ol_informix1170
Client
Client
Client
Client
SLA Group Alias
sla_sec1 SLA defined on seperate Connection Manager
sla_sec2 defined on seperate Connection Manager
sla_sec3 defined on seperate Connection Manager
ol_sec_1170
ol_sec_1170_2
SDS nodes

Figure 11-8 Multiple SLAs and SLQA groups

In the following example, the infrastructure is based on simple primary/SDS pairs. Three Connection Managers are defined to maintain one SLA each. The SLAs are combined in a group in the `sqlhosts` file. The Connection Managers are started. After that, the first Connection Manager is stopped. A subsequent connection attempt of a new client to the group alias leads to a redirection of the client to the second Connection Manager. Be aware that the client connects to the server group, not to the SLA itself. Example 11-21 shows the details.

Example 11-21 Using SLAs in server groups

```
#sqlhosts entry
ol_informix1170 onsoctcp informixva ol_informix1170
sla_sec_grp group - - i=8
sla_sec1  onsoctcp informixva sla_sec1 g=sla_sec_grp
sla_sec2  onsoctcp informixva sla_sec2 g=sla_sec_grp
sla_sec3  onsoctcp informixva sla_sec3 g=sla_sec_grp

#ucm1.config
NAME            cm1
LOG             1
LOGFILE         /tmp/cm1.log

DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_sec1    DBSERVERS=sds
}

#ucm2.config
NAME            cm2
LOG             1
LOGFILE         /tmp/cm2.log

DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_sec2   DBSERVERS=sds
}
#ucm3.config
NAME            cm3
LOG             1
LOGFILE         /tmp/cm3.log

DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_sec3   DBSERVERS=sds
}

#start the connection manager
$oncmsm -c ./ucm1.config
$oncmsm -c ./ucm2.config
$oncmsm -c ./ucm3.config

#Verification
$> onstat -g cmsm
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 2 days 20:26:10 -- 160540 Kbytes
Connection Manager Name: cm2
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_sec2/CLUSTER/mycluster       3   sla_sec2/onsoctcp   DBSERVERS=sds

Connection Manager Name: cm3
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_sec3/CLUSTER/mycluster       1   sla_sec3/onsoctcp   DBSERVERS=sds

Connection Manager Name: cm1
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_sec1/CLUSTER/mycluster       0   sla_sec1/onsoctcp   DBSERVERS=sds

Failover Configuration:
Connection Manager name        Rule                Timeout   State
cm2                            ORDER=SDS                 0   Active Arbitrator, Primary is up
cm3                            ORDER=SDS                 0   Primary is up
cm1                            ORDER=SDS                 0   Primary is up
```

```
#cm2 is the active arbitrator - will bring it down
$oncmsm -k cm2

#onstat -g cmsm output - Note cm2 is down and cm3 is active arbitrator
Failover Configuration:
Connection Manager name        Rule              Timeout   State
cm3                            ORDER=SDS               0   Active Arbitrator, Primary is up
cm1                            ORDER=SDS               0   Primary is up

#Environment
export INFORMIXSERVER=sla_sec_grp
export INFORMIXDIR=/opt/IBM/informix

#start the next client connecting to the group alias
$testapp sla_sec_grp&
$testapp sla_sec_grp&

#verification -- check the new number of connections for cm2
$> onstat -g cmsm

Connection Manager Name: cm3
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_sec3/CLUSTER/mycluster       1   sla_sec3/onsoctcp   DBSERVERS=sds

Connection Manager Name: cm1
        Hostname: informixva

        SLA                    Connections   Service/Protocol   Rule
        sla_sec1/CLUSTER/mycluster       1   sla_sec1/onsoctcp   DBSERVERS=sds
```

11.1.8 Troubleshooting

In this section, we introduce some common mistakes that can be made during the configuration and start time of the Connection Manager. These mistakes are mostly related to either an incorrect environment setup or a misunderstanding of the basic concepts.

Missing environment variables

Because the Connection Manager ONCMSM is also a client application, there are the same basic requirements in terms of environment variable settings. If the appropriate INFORMIXSQLHOSTS variable is not set, or the entry for the defined SLA is not the in current `sqlhosts` file, the ONCMSM will not start. Example 11-22 demonstrates this common error situation.

Example 11-22 Missing the SLA entry in the sqlhosts file

```
#INFORMIXSQLHOSTS is not set
$env | grep INFORMIXSQLHOSTS
```

```
INFORMIXSQLHOSTS=

#Attempt to startup oncmsm
$oncmsm -c ./ucm.config

#Log file output
$tail -f /tmp/cm1.log
21:30:01 Connection Manager name is cm1
21:30:01 Starting Connection Manager...
21:30:02 set the maximum number of file descriptors 32767 failed
21:30:02 setrlimit(RLIMIT_NOFILE, 32767) = -1, errno = 0 [cmsm_main.c:3723]
21:30:02 the current maximum number of file descriptors is 4096
21:30:02 Get server ol_informix1170 information failed, Connection Manager stopped

#Missing SLA in $INFORMIXSQLHOSTS
$cat $INFORMIXSQLHOSTS | grep sla_prim
$

#Attempt to start oncmsm
$oncmsm -c ./ucm.config
$tail -f /tmp/cm1.log
21:35:50 dbservername = ol_informix1170
21:35:50 nettype      = onsoctcp
21:35:50 hostname     = informixva
21:35:50 servicename  = ol_informix1170
21:35:50 options      =
21:35:50 listener sla_prim initializing
21:35:50 Warning: Cannot find sla_prim in INFORMIXSQLHOSTS
21:35:50 create new thread -1229378672 for ol_informix1170 [cmsm_sla.c:728]
21:35:50 switch to daemon mode
21:35:50 new daemon pid is 7518 [cmsm_main.c:3638]
21:35:50 listener sla_sec initializing
21:35:50 SLA sla_prim listener failed [cmsm_main.c:3919]
21:35:50 Error: SLA listener failed, Connection Manager shut down
```

Misunderstanding basic concepts

SLA definitions and target primary database definitions are placed together in the same `sqlhosts` file. You must be sure that there is no conflict in the naming conventions. You cannot name the SLA with the same name as the primary database server. The ONCMSM attempts to open the configured port on the local machine to instantiate a listener thread for serving incoming requests. If you define the SLA with the same name, the ONCMSM attempts to open the port of the target database server, where either the database server is on the same machine in use or the database server is on a remote machine that you cannot open.

You might find an error in the log file, as shown in Example 11-23.

Example 11-23 Do not name the SLA the same as the database server

```
$cat $INFORMIXSQLHOSTS
ol_informix1170 onsoctcp informixva ol_informix1170
ol_sec1170 onsoctcp informixva ol_sec1170
ol_sec1170_2 onsoctcp informixva ol_sec1170_2
sla_prim onsoctcp informixva sla_prim
sla_sec  onsoctcp informixva sla_sec

#Note INFORMIXSERVER (primary) and SLA name are the same
$cat ucm.config
NAME            cm1
LOG             1
LOGFILE         /tmp/cm1.log

DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     ol_informix_1170   DBSERVERS=primary
        SLA     sla_sec   DBSERVERS=sds
}

#Bring up oncmsm
$oncmsm -c ucm.config

$tail -f ucm1.log
13:13:31 dbservername = ol_informix1170
13:13:31 nettype      = onsoctcp
13:13:31 hostname     = informixva
13:13:31 servicename  = ol_informix1170
13:13:31 options      = g=g_1
13:13:31 listener ol_informix_1170 initializing
13:13:31 Warning: Cannot find ol_informix_1170 in INFORMIXSQLHOSTS
13:13:31 create new thread -1227826288 for ol_informix1170 [cmsm_sla.c:728]
13:13:31 switch to daemon mode
13:13:31 new daemon pid is 7518 [cmsm_main.c:3707]
13:13:31 listener sla_sec initializing
13:13:31 SLA ol_informix_1170 listener failed [cmsm_main.c:3988]
13:13:31 Error: SLA listener failed, Connection Manager shut down
```

Maintaining the Connection Manager with different users

In Connection Manager administration, there is one dedicated user account maintaining the ONCMSM. If you have used another user in the previous start of the ONCMSM, the subsequent start might fail because of permission problems with accessing internal maintenance files. Look at the possible error messages on the panel and in a temporarily created log file, as shown in Example 11-24. The example shows an attempt to restart a ONCMSM as with the informix user. The previous run of the ONCMSM was issued as root. Informix does not have access to the log file and an internally created file. To solve this issue, both files must be removed before reattempting to start the executable.

Example 11-24 Permission problems at startup time of the ONCMSM

```
$ONCMSM -c ONCMSM.cfg
#output on the screen :
07:21:14 cannot open logfile /tmp/file, errno = 13 [cmsm_main.c:2119]
07:21:14 Warning cannot process logfile /tmp/file
07:21:14 Connection Manager name is cm1
07:21:14 Starting Connection Manager, please check log file
/vobs/IDS11/tmp/cm1.29558.log

#output in the Logfile:
07:21:14 IBM Informix Connection Manager
07:21:14 Warning: Password Manager failed, working in trusted node
07:21:14 current max open fd is 1024
07:21:14 file /tmp/cmsm.pid.cm1 open error 13 [cmsm_main.c:1633]
07:21:14 switch to daemon mode
07:21:14 new daemon pid is 29559 [cmsm_main.c:2336]
```

11.2 Failover management in an Informix cluster environment

In previous sections, we discussed in depth the ability of the ONCMSM utility to provide load balancing for client applications based on the guidelines given by the administration. But there is another important function that the utility also can provide. The ONCMSM can also monitor the availability of the primary database server in the Informix cluster environment. In case of an outage of the primary server, an automatic failover can be done, again, depending on a defined configuration parameter. This function is the *failover arbitrator*.

In this section, we provide, in a similar structure as for the Connection Manager, an in-depth overview of the basic concepts of the failover management functionality provided by the ONCMSM. The new ONCMSM provides failover management for all the clusters it manages.

Be aware that the failover management and the connection management functionality is provided by the same utility, that is, ONCMSM. There are the same requirements in terms of product installation as we discussed in 11.1.1, “Installation of the Connection Manager” on page 413, even if you only want to use the failover arbitrator in a stand-alone server.

11.2.1 Failover configuration

The failover arbitrator requires, in addition to the necessary configuration parameters for the Connection Manager, the existence of one other parameter. This parameter goes in the configuration file and is highlighted by the FOC keyword (present within a CLUSTER section). If you have already created a configuration file for the Connection Manager functionality, simply add the new parameter to that file. The change of the configuration is recognized by the next restart of the executable.

If you only want to use the failover arbitrator, you have be careful when using the FOC (failover configuration) parameter. Apply some defaults to the SLA definition. These defaults only come into play if the client connects to the SLA. If the clients remain connected to the database server definition, the Connection Manager is not involved and remains in dummy functionality.

Using the configuration file

Here we discuss the configuration of the failover arbitrator. The failover arbitrator requires the existence of the following parameter in the configuration file within a CLUSTER section:

```
FOC ORDER=<failover configuration guidelines> TIMEOUT=<time-out value>
RETRY=<number of retries>
```

The following values are allowed for the FOC guideline settings:

- SDS: The new primary is chosen from the list of the available SDS servers in the cluster.
- RSS: The new primary is chosen from the list of the available RSS servers in the cluster.
- HDR: The new primary is the HDR secondary server.
- Any database server name in the cluster.

Different values can be separated with a “,”, indicating the order in which the server combinations are tried out, for example, SDS,RSS will first try failover with the SDS servers and then try the RSS servers. Values can be assigned the same priority by enclosing them in parenthesis, for example, (SDS,RSS).

The configurations of SDS, HDR, and RSS do not specify a certain server, they specify database servers belonging a configured type. At the primary database server outage time, one of the available database servers in the cluster, the same as the FOC specified type, is promoted to the primary.

We have compiled a list of possible settings of the FOC parameter. Example 11-25 shows the selection of the settings. Be aware that the use of the name “c_sec” in this example list identifies a database server by the database server name and not by the type.

Example 11-25 Possible settings to define a failover strategy in a configuration file

```
FOC ORDER=SDS TIMEOUT=0
FOC ORDER=RSS TIMEOUT=60
FOC ORDER=HDR TIMEOUT=40
FOC ORDER=SDS,HDR TIMEOUT=10
FOC ORDER=c_sec TIMEOUT=0
FOC ORDER=c_sec,(SDS,HDR) TIMEOUT=100000

#default setting if nothing is specified at startup time
FOC ORDER=SDS+HDR+RSS TIMEOUT=0
```

The specification of a timeout value other than 0 means that the arbitrator will wait, when there is an outage of the primary database server, the specified amount of seconds before attempting to apply the configured failover strategy.

Be careful when setting the specification of the timeout value. For example, say there is a simple HDR solution and the ONCMSM is started with a low timeout value monitoring the server. If you bring down the primary, the failover arbitrator comes in place immediately and, depending on your DRAUTO settings, you end up with two independent primary servers.

Conversely, if there is a real outage of the primary, the expectation of an immediate switchover to participate in the benefits of the cluster is quite understandable. A higher value could block current and new incoming connections for a longer time, which should be certainly be avoided.

11.2.2 Common simple failover scenarios

In this section, we illustrate, based on some sample failover scenarios, the abilities of the failover arbitrator. The examples describe the environment we use and the setup of the ONCMSM in regards to the failover settings. The failover is in all examples initiated by running **`onmode -ky`** on the primary server in the cluster.

Simple SDS environment

Here we have an SDS infrastructure with a primary and two attached SDS servers in place. In a failover situation, we promote one of the SDS servers automatically to be the new primary server. The other SDS server remain as an SDS server. Figure 11-9 shows the cluster environment.

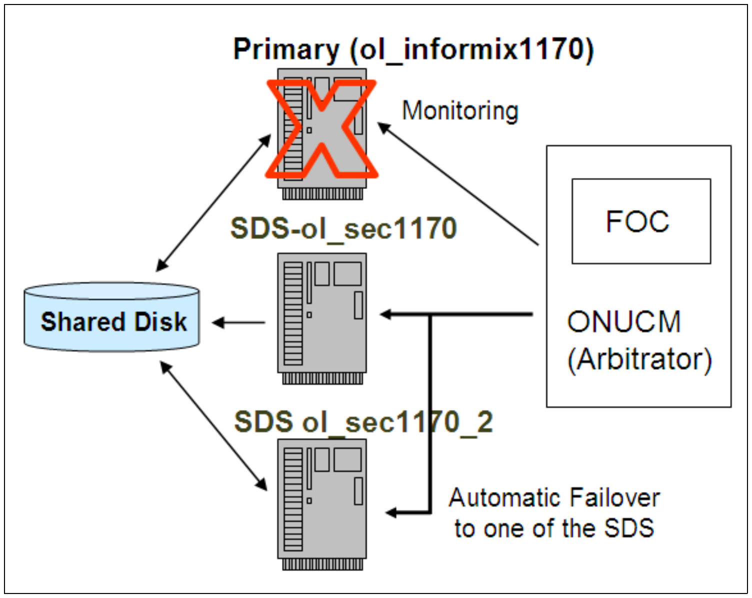


Figure 11-9   Simple SDS failover

Example 11-26 shows the steps for the setup of the arbitrator. You have to specify the -f option using the command-line configuration, and running **`onstat -g cmsm`** shows that the FOC is in place. We shut down the primary. The chosen SDS promoted to be the next primary finishes the fast recovery phase and changes the server type. The remaining secondary only changes the attached primary name.

Example 11-26   Promoting an SDS to a primary in case of a failover

```
#environment
export INFORMIXDIR=/opt/IBM/informix
export INFORMIXSERVER=ol_informix1170

#Output of ucm.config file
$cat ucm.config
NAME            cm1
LOG             1
LOGFILE         /tmp/cm1.log
```

```
DEBUG   1
CLUSTER mycluster
{
        INFORMIXSERVER  ol_informix1170
        SLA     sla_prim   DBSERVERS=primary
        SLA     sla_sec   DBSERVERS=sds
          FOC ORDER=SDS
}

#start the Failover Arbitrator
$> oncmsm -c ./ucm1.config
#Check the settings
$> onstat -g cmsm
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 2 days 22:27:12 -- 160540 Kbytes
Connection Manager Name: cm1
        Hostname: informixva

        SLA                     Connections   Service/Protocol    Rule
        sla_prim/CLUSTER/mycluster       1   sla_prim/onsoctcp   DBSERVERS=primary
        sla_sec/CLUSTER/mycluster        0   sla_sec/onsoctcp    DBSERVERS=sds
        sla_any/SERVERSET/mygrid         0   sla_any/onsoctcp
                                        DBSERVERS=utm_group_1,utm_group_2,utm_group_3  POLICY=LATENCY

Failover Configuration:
Connection Manager name        Rule               Timeout   State
cm1                            ORDER=SDS                0


#onmode -ky for the primary


#Check the Logfiles for CM
13:34:49 Cluster mycluster Arbitrator FOC string = ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1
13:34:49 Cluster mycluster Arbitrator setting primary name = ol_sec1170 [cmsm_arb.c:432]
13:34:49 FOC[0] = SDS
13:34:49 FOC[1] = HDR
13:34:49 FOC[2] = RSS
13:34:49 FOC TIMEOUT = 0
13:34:49 FOC RETRY = 1
13:34:49 get ol_sec1170 CLUST_CHG event 1:3[CONNECT]
ol_sec1170_2|SDS|onsoctcp|informixva|ol_sec1170_2|g=g_1  [cmsm_er.ec:1475]
13:34:49 get ol_sec1170 CLUST_CHG event 1:2[DROP] ol_sec1170|SDS| [cmsm_er.ec:1475]
13:34:49 get ol_sec1170 CLUST_CHG event 1:5[NEWPRI] ol_sec1170 [cmsm_er.ec:1475]
13:34:50 Arbitrator make primary on node = ol_sec1170 successful

#Online.log file of the force failed primary server
13:34:42  IBM Informix Dynamic Server Stopped.

#Log file of the secondary
13:34:46  DR: Turned off on secondary server
13:34:46  Logical Recovery has reached the transaction cleanup phase.
13:34:46  Checkpoint Completed:  duration was 0 seconds.
13:34:46  Fri Feb 25 - loguniq 12, logpos 0x9be018, timestamp: 0xfb17a Interval: 254

13:34:46  Maximum server connections 4
13:34:46  Checkpoint Statistics - Avg. Txn Block Time 0.000, # Txns blocked 0, Plog used 3, Llog used
1
...
13:34:47  B-tree scanners enabled.
13:34:48  On-Line Mode
13:34:49  SDS Server ol_sec1170_2 - state is now connected
13:34:49  DR: Reservation of the last logical log for log backup turned on
```

```
#onstat on the secondary which has taken over as the primary
$onstat -g sds
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line -- Up 2 days 22:41:34 -- 152348 Kbytes

Local server type: Primary
Number of SDS servers:1

SDS server information

SDS srv      SDS srv      Connection           Last LPG sent        Supports
name         status       status               (log id,page)        Proxy Writes
ol_sec1170_2 Active       Connected                 12,2554         N
```

Promoting an SDS server in a mixed cluster environment

Because the simple infrastructures, such as a pure HDR or a pure RSS environment, behave similarly, we do not discuss them here in detail. Rather, we describe the failover scenario for a heterogeneous cluster environment with a primary database server that has SDS, RSS, and HDR secondary servers attached. The reason we are considering this scenario is that, depending on the chosen failover strategy, the failover behaves differently and may impact some of the manual administration activities. Figure 11-10 shows the cluster with much of the available Informix HA solution involved. There are two major strategies that we can follow in case the primary has seen an outage: failover to an SDS and failover to an RSS.

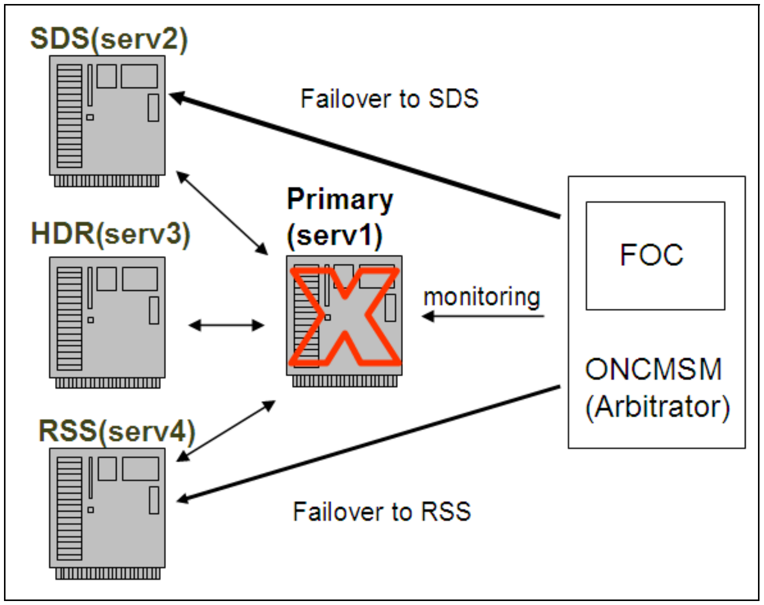


Figure 11-10 Complex Informix cluster infrastructure with possible failover strategies

The simplest failover case in our infrastructure is the failover to an SDS. Similarly, as in the previous example, we show the steps in the beginning of the configuration of the failover as a practical use case. Example 11-27 shows all the necessary details. After the primary has failed, the SDS is promoted to the new primary. For the success of the failover, we check the other participating server types as well. The RSS shows the source server in the **onstat -g rss** output. You can see that the name is noted there with the serv2. The HDR secondary indicates the primary server name to be connected within the **onstat -g dri** output, again the new primary server serv2. The new primary indicates its status at in the **onstat -g rss** output as well. The banner of the output also indicates that the server is now an HDR primary.

Example 11-27 Promoting an SDS server to a primary using the arbitrator

```
#environment
export INFORMIXDIR=/opt/IBM/informix
export INFORMIXSERVER=serv1

$cat $INFORMIXSQLHOSTS
utm_group_1 group - - i=1
serv1 onsoctcp hal14.lenexa.ibm.com 14001 g=utm_group_1
serv3 onsoctcp hal14.lenexa.ibm.com 14003 g=utm_group_1
serv4 onsoctcp hal14.lenexa.ibm.com 14004 g=utm_group_1
serv2 onsoctcp hal14.lenexa.ibm.com 14002 g=utm_group_1
sla_cluprimonsoctcpinformixva sla_cluprim
sla_cluothonsoctcpinformixvasla_cluoth

$cat config.cluster
NAME           cmclust
LOG            1
LOGFILE        /tmp/clu.log

DEBUG   1
CLUSTER halclust
{
        INFORMIXSERVER  serv1
        SLA     sla_cluprim   DBSERVERS=primary
        SLA     sla_cluoth   DBSERVERS=sds,rss
        FOCorder=sds
}

#onstat -g cmsm at the primary
$ onstat -g cmsm
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 2 days 19:04:51 -- 160540
Kbytes
Connection Manager Name: cmclust
        Hostname: informixva

        SLA                    Connections   Service/Protocol       Rule
        sla_cluprim/CLUSTER/halclust     0   sla_cluprim/onsoctcp   DBSERVERS=primary
        sla_cluoth/CLUSTER/halclust      0   sla_cluoth/onsoctcp    DBSERVERS=sds,rss

Failover Configuration:
Connection Manager name        Rule               Timeout   State
cmclust                        order=sds                0   Active Arbitrator, Primary is up

#Look at the onstat -g dri on the RSS node to see the server it is paired with
$onstat -g rss


IBM Informix Dynamic Server Version 11.70.UC2 -- Updatable (RSS) -- Up 2 days 19:01:23 -- 176924
Kbytes

Local server type: RSS
Server Status : Active
Source server name: serv1
Connection status: Connected
Last log page received(log id,page): 10,2268

#initiate the failover with an onmode -ky (serv1)
$onmode -ky
#
#Look at the logfile of the UCM .
09:57:28 Connection Manager disconnected from serv1
09:57:28 Arbitrator detected down server, svr=serv1, type=1 [cmsm_arb.c:842]
09:57:28 CLuster halclust Arbitrator failover to serv2, waiting 300 seconds for confirmation
[cmsm_arb.c:1169]
09:57:29 Arbitrator connected to failover node = serv2 [cmsm_server_arb.ec:193]
09:57:32 Arbitrator make primary on node = serv2 successful

#Look at onstat -g rss on the RSS node.
$> onstat -g rss

IBM Informix Dynamic Server Version 11.70.UC2 -- Updatable (RSS) -- Up 2 days 19:12:30 -- 217192
Kbytes

Local server type: RSS
Server Status : Active
Source server name: serv2
Connection status: Connected
Last log page received(log id,page): 11,48

#The HDR secondary -- Paired server is the previous SDS server and the new
primary
$onstat -g dri
IBM Informix Dynamic Server Version 11.70.UC2 -- Updatable (Sec) -- Up 2 days 19:06:01 -- 200808
Kbytes

Data Replication at 0x4b4531a0:
  Type          State       Paired server       Last DR CKPT (id/pg)    Supports Proxy Writes
  HDR Secondary  on          serv2                       11 / 2          Y

  DRINTERVAL   30
  DRTIMEOUT    30
  DRAUTO       3
  DRLOSTFOUND  /work/nilesho/testnode/sqldist/etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0
  Backlog      0

#onstat -g rss on new primary (serv2)
sh-3.2$ onstat -g rss
```

```
IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 2 days 19:25:27 -- 200808
Kbytes

Local server type: Primary
Index page logging status: Enabled
Index page logging was enabled at: 2011/02/25 13:50:13
Number of RSS servers: 1

RSS Server information:

RSS Srv      RSS Srv      Connection     Next LPG to send       Supports
name         status       status           (log id,page)        Proxy Writes
serv4        Active       Connected              11,51          Y
```

Promoting an RSS in a mixed cluster environment

The previous example was a simple failover case, because the SDS and the primary shared the same disk environment. Now we use the same Informix cluster infrastructure and set up the arbitrator to promote an RSS database server to a primary in the failover case. We again refer to the sample Informix cluster solution described in Figure 11-10 on page 453.

There are some differences in the handling of the remaining SDS secondary servers when you promote an RSS. The promotion of an HDR, which we do not discuss here, is quite similar. You can see the differences in comparison with the previous scenario in the setup and what additions have to be manually done on the administration side for the SDS illustrated in Example 11-28.

After the primary in the Informix cluster fails, the RSS server is promoted to the primary as expected. The HDR secondary switches the primary to the serv4 server as expected, because this is now the primary. The SDS server is brought down at the request of the new primary, which is different than in the SDS failover case. This action is shown in the snippet of the log file in the example. The manual attempt to start the SDS server again will fail because all servers are working with the same copy of the dbspaces, but the reserved page 0 is different in the RSS in terms of the disk owner than on the previous cluster primary which also shared the SDS. You have to switch the ROOTPATH in the `onconfig` from the failed primary to the new primary. If you are using relative path names, you have to switch to the directory of the previous RSS (now primary) database server for startup.

Example 11-28 Promoting an RSS to the new primary using the arbitrator

```
#config file for ucm
$cat config.cluster
NAME           cmclust
LOG            1
LOGFILE        /tmp/clu.log

DEBUG   1
CLUSTER halclust
```

```
{
        INFORMIXSERVER  serv1
        SLA     sla_cluprim   DBSERVERS=primary
        SLA     sla_cluoth   DBSERVERS=sds,rss
        FOCorder=rss
}

#Start UCM and make sure it is registered properly
$oncmsm -c ./config.cluster

#monitor the setup at the prrimary
$ onstat -g cmsm

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 00:25:22 -- 160540 Kbytes
Connection Manager Name: cmclust
        Hostname: informixva

        SLA                    Connections   Service/Protocol       Rule
        sla_cluprim/CLUSTER/halclust     0   sla_cluprim/onsoctcp   DBSERVERS=primary
        sla_cluoth/CLUSTER/halclust      0   sla_cluoth/onsoctcp    DBSERVERS=sds,rss

Failover Configuration:
Connection Manager name        Rule               Timeout   State
cmclust                        order=rss                0   Active Arbitrator, Primary is up

sh-3.2$ onstat -g dri

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 00:29:32 -- 160540 Kbytes

Data Replication at 0x4b4511a0:
  Type           State        Paired server        Last DR CKPT (id/pg)    Supports Proxy Writes
  primary        on           serv3                         4 / 18         NA

sh-3.2$ onstat -g sds

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 00:30:19 -- 160540 Kbytes

Local server type: Primary
Number of SDS servers:1

SDS server information

SDS srv      SDS srv      Connection        Last LPG sent        Supports
name         status       status            (log id,page)        Proxy Writes
serv2        Active       Connected               4,21           Y

sh-3.2$ onstat -g rss

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 00:30:21 -- 160540 Kbytes

Local server type: Primary
Index page logging status: Enabled
Index page logging was enabled at: 2011/02/28 13:03:44
Number of RSS servers: 1

RSS Server information:

RSS Srv      RSS Srv      Connection     Next LPG to send        Supports
name         status       status           (log id,page)         Proxy Writes
serv4        Active       Connected               4,22           Y
```

```
#Bring down the primary
$onmode -ky

#Check the server types in the modified Cluster environment
#HDR secondary server -- changed the primary name to c_rss the prev RSS
IBM Informix Dynamic Server Version 11.50.UC1 -- Updatable (Sec) -- Up 00:13:59 -- 51924 Kbytes

Data Replication:
  Type           State        Paired server        Last DR CKPT (id/pg)
  HDR Secondary  on           c_rss                         8 / 1

  DRINTERVAL   30
  DRTIMEOUT    30
  DRAUTO       0
  DRLOSTFOUND  /usr/informix/etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0

#The RSS (ser4) -- should be now a primary server
$> onstat -

IBM Informix Dynamic Server Version 11.50.UC1 -- On-Line (Prim) -- Up 00:16:19 -- 51924 Kbytes
#The HDR secondary is connected to the new primary (serv4)
$onstat -g dri

IBM Informix Dynamic Server Version 11.70.UC2 -- Updatable (Sec) -- Up 00:32:43 -- 152348 Kbytes

Data Replication at 0x4b4531a0:
  Type           State        Paired server        Last DR CKPT (id/pg)    Supports Proxy Writes
  HDR Secondary  on           serv4                         4 / 33         Y

  DRINTERVAL   30
  DRTIMEOUT    30
  DRAUTO       3
  DRLOSTFOUND  /work/nilesho/testnode/sqldist/etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0
  Backlog      0

#Check the SDS (serv2) -- currently down
$ onstat -
shared memory not initialized for INFORMIXSERVER 'serv2'

#Check the Logfile
13:34:59  SMX thread is exiting because of network error code -25582
13:34:59  SMX thread is exiting because of network error code -25582
13:35:00  Updates from secondary currently not allowed
13:35:00  Updates from secondary currently not allowed
13:35:00  SDS: Lost connection to serv1
13:35:09  SDS: Received Shutdown request from serv4, Reason: DR Secondary switched to DR Primary role
13:35:09  DR:Shutting down the server.
13:35:09  Updates from secondary allowed
13:35:15  IBM Informix Dynamic Server Stopped.

#Try to bring up the secondary again -- most likely hung
$oninit
#Notice errors in the log file
#Change root path to point to serv4’s ROOTPATH
#Try bringing it up again
$oninit
```

```
sh-3.2$ onstat -g sds

IBM Informix Dynamic Server Version 11.70.UC2 -- On-Line (Prim) -- Up 00:51:50 -- 160540 Kbytes

Local server type: Primary
Number of SDS servers:1

SDS server information

SDS srv      SDS srv      Connection        Last LPG sent        Supports
name         status       status            (log id,page)        Proxy Writes
serv2        Active       Connected              4,42            Y
```

11.2.3 Troubleshooting and common mistakes

We now want to discuss some common configuration mistakes.

Permission problems

To support a successful failover in case of a database server outage, the ONCMSM executable has to be executed as by the informix user. This requirement results from the need to access the sysadmin system database in case of failover. The failover itself is done by the execution of the task function provided by the sysadmin database. In a default database server state, only the informix user has the permissions to access to this database. Example 11-29 shows the failed attempt to try the failover when the ONCMSM was started by root. The -387 error returned by the **`execute function task()`** call indicates a connect permission problem to the sysadmin database.

Example 11-29 Failover failed because database permissions are missing

```
07:22:15 fetch sysrepstats_cursor SQLCODE = (-25582,0,) [cmsm_server.ec:785]
07:22:15 register event failed, sqlcode = (-1811,0,) [cmsm_server.ec:695]
07:22:15 Connection Manager disconnect from ol_informix1170
07:22:15 Arbitrator detected down server, svr=ol_informix1170, type=256 [cmsm_arb.c:821]
07:22:15 Arbitrator detect total sec count = 2, disconnected = 0 [cmsm_arb.c:900]
07:22:15 Arbitrator start failover logic now... [cmsm_arb.c:924]
07:22:15 Arbitrator try failover on specific aliases [cmsm_arb.c:696]
07:22:15 Arbitrator found failover node = ol_sec1170 [cmsm_server_arb.ec:143]
07:22:15 Arbitrator CONNECT to sysmaster@ol_sec1170|onsoctcp|Localhost|30670 AS
[cmsm_server_arb.ec:96]
07:22:16 Arbitrator CONNECT to sysmaster@ol_sec1170|onsoctcp|Localhost|30670 SQLCODE = 0
[cmsm_server_arb.ec:110]
07:22:16 Arbitrator connected to failover node = ol_sec1170
[cmsm_server_arb.ec:155]
07:22:16 Arbitrator prepare str=EXECUTE FUNCTION sysadmin:task("ha make primary force",
"ol_sec1170"), SQLCODE=-387 [cmsm_server_arb.ec:186]
07:22:16 Arbitrator can not find any secondary nodes for failover.
```

Definition of a server outside of the cluster in the FOC

You may specify the name of a specific database server in the FOC parameter instead of a server type, such as HDR or SDS, to define a failover strategy. This database server has to be part of the same cluster. You should not specify any database server name that is part of a different cluster. In that case, the failover will not take place. Example 11-30 illustrates what happens. As a given, we have two separate clusters. One is an SDS environment with a primary server and one is an SDS server attached to the primary. There is another cluster environment with two HDR servers. There is an ONCMSM that has started monitoring the SDS environment. The FOC specifies the HDR secondary for failover. In case of the failover, the arbitrator recognizes that there is one remaining SDS database server. This server is compared with the given FOC strategy, which does not match. Therefore, no failover takes place.

Example 11-30 FOC does not specify a server of the same cluster

```
#sqlhosts
#database server
c_prim onsoctcp localhost 1111
c_sec onsoctcp localhost 1112

c_hdr_prim onsoctcp localhost 1113
c_hdr_sec onsoctcp localhost 1114
#SLA defintion
oltp1 onsoctcp localhost 1115

#ONCMSM.cfg file
NAME cm1
LOGFILE /tmp/logfile
CLUSTER myclust
{
SLA oltp1 DBSERVERS=SDS
FOC ORDER=c_hdr_sec
}
DEBUG 1

#Output from the ONCMSM logfile in case of the failover attempt
06:30:30 CONNECT to sysmaster@c_prim|onsoctcp|Localhost|2300 SQLCODE = (-908,107,c_prim)
[cmsm_server.ec:85]
06:30:30 Arbitrator detected down server, svr=c_prim, type=256 [cmsm_arb.c:821]
06:30:30 Arbitrator detect total sec count = 1, disconnected = 0 [cmsm_arb.c:900]
06:30:30 Arbitrator start failover logic now... [cmsm_arb.c:924]
06:30:30 Arbitrator try failover on specific aliases [cmsm_arb.c:696]
06:30:30 Arbitrator can not find any secondary nodes for failover.
06:30:30 CONNECT to sysmaster@c_prim|onsoctcp|Localhost|2300   [cmsm_server.ec:71]
06:30:30 CONNECT to sysmaster@c_prim|onsoctcp|Localhost|2300 SQLCODE = (-908,107,c_prim)
[cmsm_server.ec:85]
```

12

Informix in cloud and virtualization environments

In this chapter, we discuss certain virtualization and cloud usage scenarios, why you need these types of usage models, what capabilities in Informix help with these environments, and the steps to set up some of these models. Virtualization is a key foundation in cloud models and we discuss Informix support and certification on various virtualization platforms. We also talk about the Informix Virtual Appliance, which the prebuilt and preconfigured virtual machine that can be downloaded and run easily to create a quick demonstration or testing with Informix.

12.1 Informix virtualization support

There are several virtualization platforms that offer server virtualization capabilities. Informix software is certified on several virtualization platforms. Table 12-1 lists the virtualization platforms that Informix supports.

Table 12-1 Virtualization platform support

Hypervisor	Architecture	Minimum Informix level
VMware ESXi 4.0 / VMware vSphere 4.0	x86/x64System listed on ESX HCL	11.50.xC1 and higher
Microsoft WIndows 2008 Hyper-V R1/R2	x64 System with INTEL-VT or AMD-V	11.50.xC6 and higher
IBM z/VM® V5.2, V5.3, V5.4, and V6.1	IBM System z®	11.50.xC1 and higher
IBM Power VM	IBM System p®	11.50.xC1 and higher
Red Hat RLHEL 5.0 Xen	x64 System with INTEL-VT or AMD-V	11.50.xC1 and higher
SUSE Linux Enterprise Server (SLES) 10 Xen HVM	x64 System with INTEL-VT or AMD-V	11.50 versions only (no 11.70 support)
Solaris Zones	x64	11.50.xC1 and higher
HP-UX Virtual Partitions (vPars)	HP Integrity Servers	11.50.xC1 and higher
HP Integrity VM	HP Integrity Servers	11.50.xC1 and higher

For the complete list of guest operation systems at the individual Hypervisor architecture level and the latest updates, go to the following address:

http://www.ibm.com/developerworks/wikis/display/im/IDS+virtualization+support

12.2 Using the Informix Virtual Appliance

The Informix Virtual Appliance is a virtual machine that consists of the operating system and the full Informix software stack, all downloadable as a single unit, that can be used to create quick test scenarios and see how things work. Most of the examples created in this book used the virtual appliance and then customized configurations to our requirements. In this section, we discuss how to download the appliance, set it up, and start using it.

Embedding IBM Informix, SG24-7666 gives information about how to create virtual machines using VMware Workstation technology.

12.2.1 Downloading VMware player or VMware Workstation

The Informix appliance can be run using either the VMware player or VMware Workstation software. The virtual appliance also works with VMware Fusion on MAC OS X, and VMware Server. You can download the VMware player from the following address:

`http://www.vmware.com/products/player/`

After the product has been downloaded and installed, the next step is to download the Informix Virtual Appliance.

12.2.2 Downloading and extracting the Informix Virtual Appliance

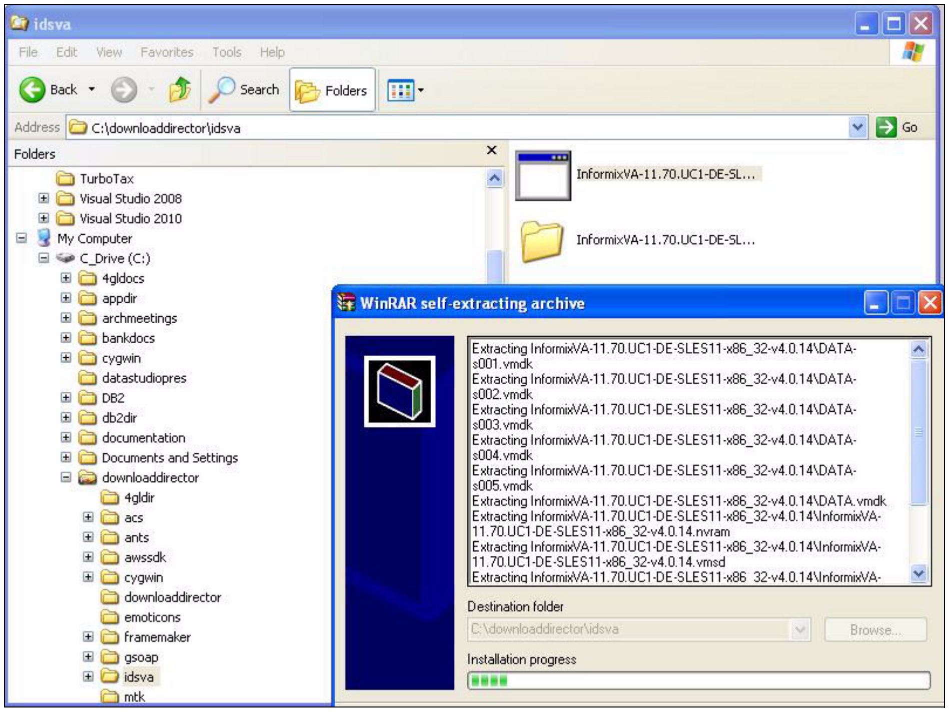
The Informix Virtual Appliance can be downloaded from the following address:

`https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-informixfpd&S_PKG=dl&lang=en_US`

You need to enter your IBM ID and password to download the Informix Virtual Appliance.

The appliance image comes as a self extracting executable or a UNIX compressed tar file and comes in both 32-bit and 64-bit formats. For this book, we use the self extracting 32-bit version, which comes in a file named `InformixVA-11.70.UC1-DE-SLES11-x86_32.exe`.

After downloading the virtual appliance, run the file to extract the pieces of the virtual image to the directory that you choose. Figure 12-1 shows the details.



```
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA-
s001.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA-
s002.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA-
s003.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA-
s004.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA-
s005.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\DATA.vmdk
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\InformixVA-
11.70.UC1-DE-SLES11-x86_32-v4.0.14.nvram
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\InformixVA-
11.70.UC1-DE-SLES11-x86_32-v4.0.14.vmsd
Extracting InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14\InformixVA-
```

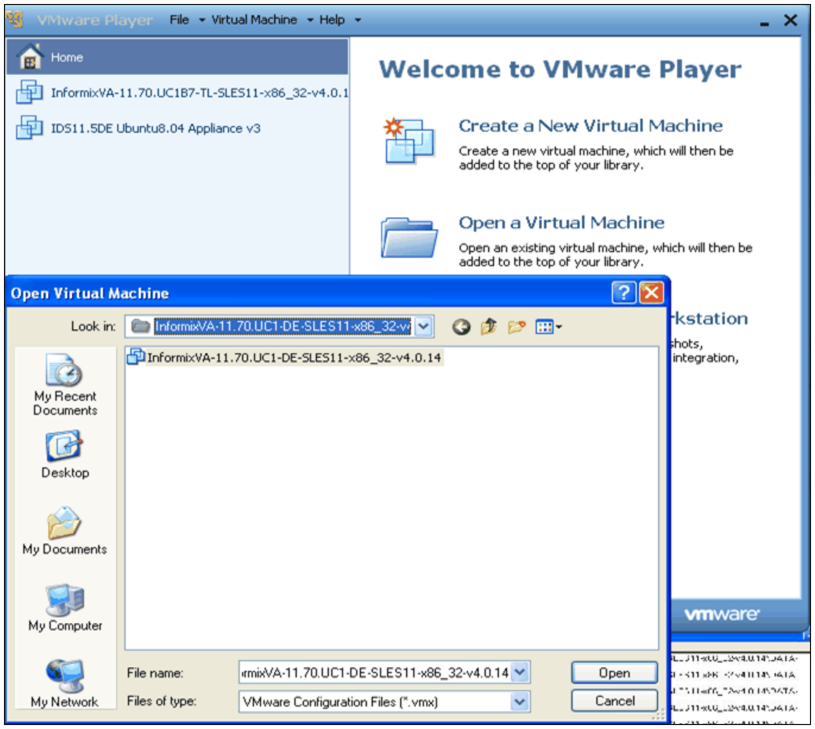
Destination folder: C:\downloaddirector\idsva

Installation progress

Figure 12-1 Extracting the downloaded image

12.2.3 Launching and using the Informix Virtual Appliance

To use the Informix Virtual Appliance, first launch the VMware player, select **Open a Virtual Machine**, select the image from the directory where it was extracted, and click **Open**. See Figure 12-2.



Create a new virtual machine, which will then be added to the top of your library.

Open an existing virtual machine, which will then be added to the top of your library.

Look in: InformixVA-11.70.UC1-DE-SLES11-x86_32-v

InformixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14

File name: rmixVA-11.70.UC1-DE-SLES11-x86_32-v4.0.14

Files of type: VMware Configuration Files (*.vmx)

Figure 12-2 Importing an extracted virtual machine

If this is the first launch of the virtual appliance, it will start and ask for the root ID and password (both are set to “root” in the appliance). Then it prompts you to accept license agreement and then prompts you for the login user name and password. For the appliance in this book, we use “informix” for both the user name and password. This brings up the initial window shown in Figure 12-3. Opening a terminal window on the virtual machine and running **`onstat`** will show that the Informix server is already up.

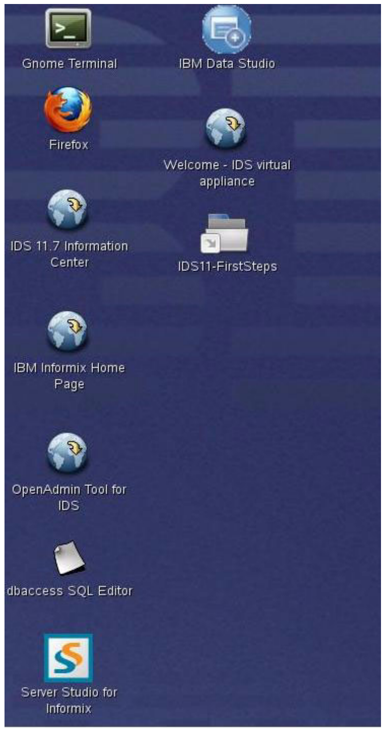


Figure 12-3 Initial virtual appliance window

12.2.4 Demonstration using the virtual appliance

The Informix Virtual Appliance provides a cluster demonstration, named *createC1Cluster*, that is located in the `$INFORMIXDIR/bin` directory.

Run **`./createC1Cluster`** at a terminal prompt to start the demonstration (Example 12-1). Running this command sets up a cluster with one primary, one HDR secondary server, two RSS servers, and two SDS servers. It creates the necessary configuration files and starts the different servers.

Example 12-1 Running the cluster demonstration

```
$cd $INFORMIXDIR/bin
$./createC1Cluster
```

You can then launch the OpenAdmin Tool (OAT) to look at the cluster configuration. To launch OAT, click the OpenAdmin Tool for IDS icon. In the OAT's initial window, select the server group **IDS11Cluster** from the OAT Group, click **Get Servers**, and then select **c1primary@localhost** as the server to connect. The server details appear and you can connect to the server by clicking **Login**, as shown in Figure 12-4.

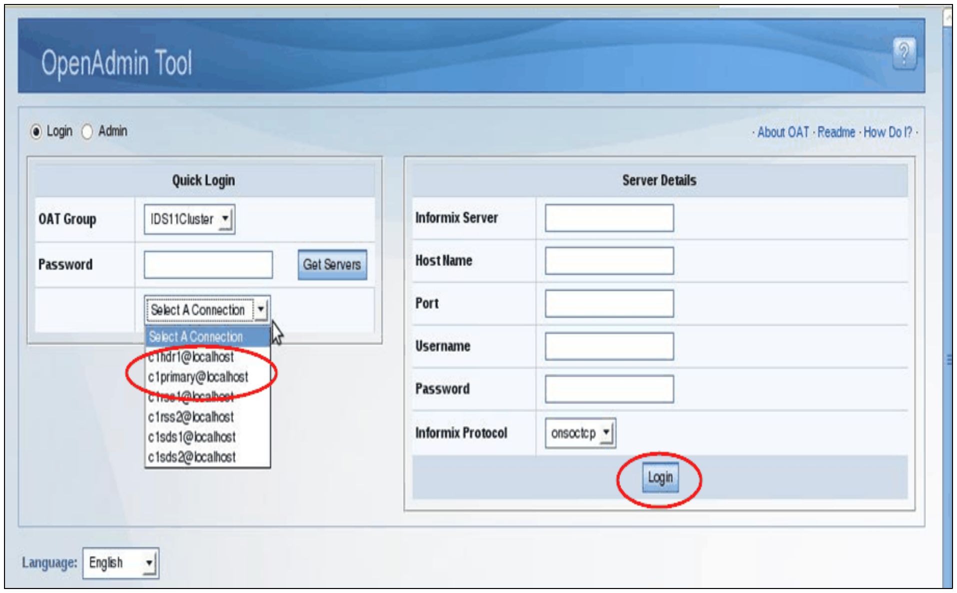


Figure 12-4 Using OAT to connect to the cluster servers

After you are connected to the server, select **Replication** → **Clusters**. You might need to download the Adobe Flash plug-in to work with the cluster. Use the following steps to enable the Flash plug-in:

1. Go to `http://get.adobe.com/flashplayer/`.
2. Download the `install_flash_player_10_linux.tar.gz` for Linux.

This step downloads the plug-in's tar.gz file into the Informix home's Downloads subdirectory:

```
informix@informixva[demo_on]:~/Downloads$ ls ~informix/Downloads
install_flash_player_10_linux.tar.gz
```

3. Extract the file using gunzip and tar:

```
informix@informixva[demo_on]:~/Downloads$ gunzip
install_flash_player_10_linux.tar.gz
informix@informixva[demo_on]:~/Downloads$ tar xvf
install_flash_player_10_linux.tar
```

4. After the file is extracted, go to the `.mozilla` directory in the informix home directory and create a `plugins` directory there.
5. Copy the `*.so` file over to the `plugins` directory:

```
informix@informixva[demo_on]:~/Downloads$ cd ~informix/.mozilla
informix@informixva[demo_on]:~/.mozilla$ ls
extensions  firefox
informix@informixva[demo_on]:~/.mozilla$ mkdir plugins
informix@informixva[demo_on]:~/.mozilla$ cp ~informix/Downloads/*.so
plugins
```

6. Exit Mozilla Firefox and relaunch it.
7. Go back to the Clusters menu. You should not see the plug-in required note that you saw before.

8. Select **Clusters** from the Replication menu. In the window that opens, click **Find Clusters** at the top and OAT starts discovering clusters. OAT presents you with the cluster information, as shown in Figure 12-5.

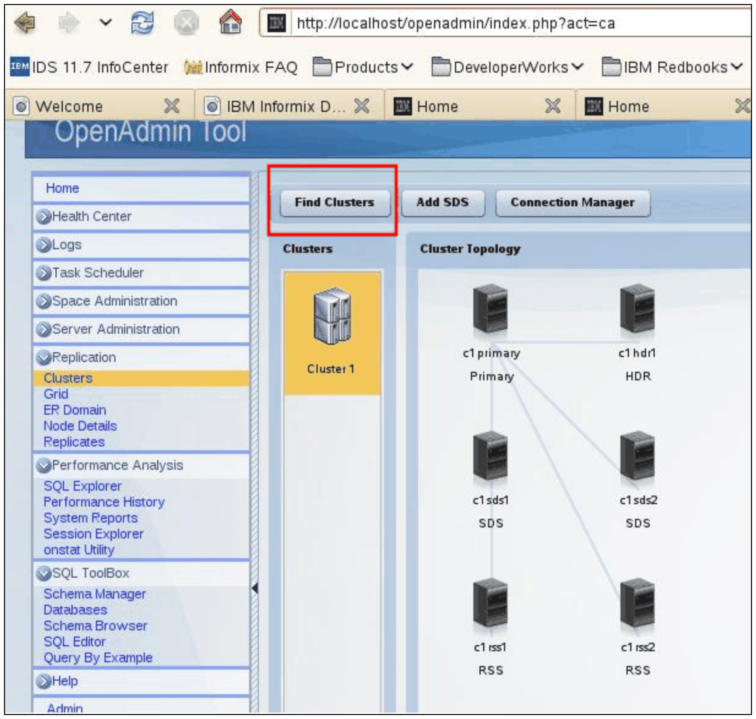


Figure 12-5 Cluster discovery in OAT

12.2.5 Virtual appliances

The Informix Virtual Appliance provides you an easy to use Informix product that can be used to understand the product and to develop and test applications. You can use tools such as VMware Workstation to build your own virtual machines with Informix servers customized to suit your requirements.

12.3 Informix in the public cloud

We now describe how to deploy Informix in IBM Smart Business Development and Test on the IBM Cloud and the Amazon EC2 public clouds.

These are infrastructure as a service (IaaS) cloud products where you can rapidly deploy an Informix server on a virtual machine instance supporting any workload type. Running Informix in these clouds can also provide cost benefits, because the instances can be provisioned on demand and the charges are determined based on usage.

12.3.1 Using Informix on IBM Smart Business Development and Test on the IBM Cloud

In this section, we provide details about deploying Informix on IBM Smart Business Development and Test on the IBM Cloud, which is an infrastructure as a service (IaaS) cloud product well suited for the development and testing of applications. It offers dynamic provisioning of enterprise-class virtual server environments, which can be used for secure and rapid deployment of Informix on virtual machines. It also provides a self-service portal for account management and environment configuration.

The example presented in this section deploys a pre-configured instance of IBM Informix Developer Edition V11.50 (a development version of the Enterprise Edition) on Red Hat Enterprise Linux 5.4 (32-bit) on IBM Cloud.

Creating an instance

You must register and set up an account with IBM before you can use IBM Smart Business Development and Test on the IBM Cloud. Register an account by going to `http://www.ibm.com`, and then sign in to the IBM Smart Business Development and Test portal at the following address:

`https://www-147.ibm.com/cloud/enterprise/dashboard`

In the following text, we refer to IBM Smart Business Development and Test on the IBM Cloud as “IBM Cloud”. Figure 12-6 shows the IBM Cloud portal.

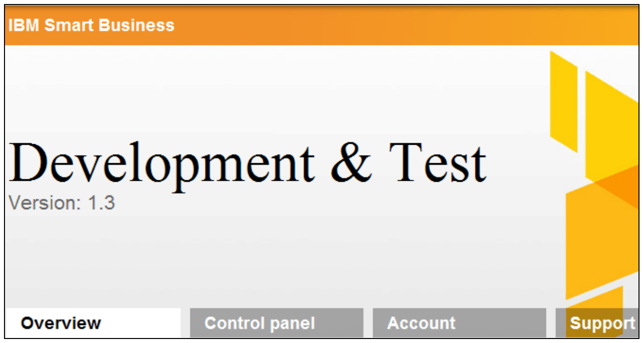


Figure 12-6 IBM Cloud portal

After signing in to the IBM Cloud, go to the Control Panel tab to start creating an instance based on the available Informix public image. Figure 12-7 shows an overview of the steps involved in creating an instance from the Control Panel.

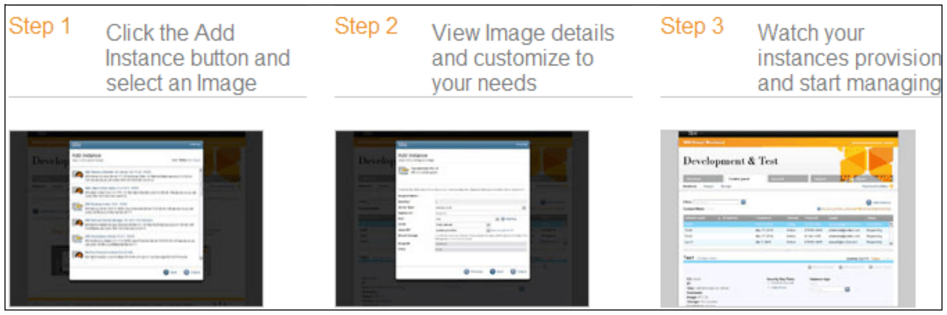
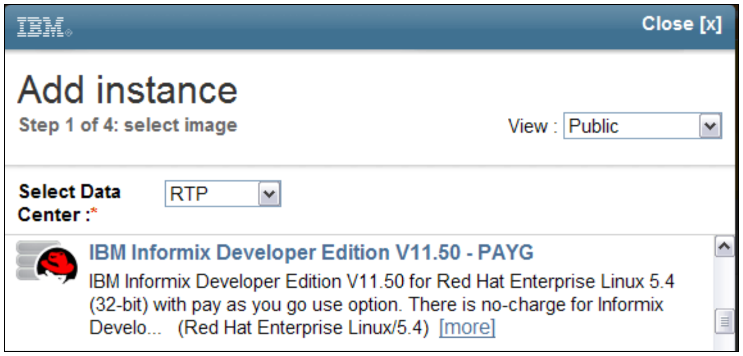

Step 1
Click the Add Instance button and select an Image
Step 2
View Image details and customize to your needs
Step 3
Watch your instances provision and start managing

Figure 12-7 Control Panel

Here we walk through these steps:

1. Click the **Add Instance** button and select an image.

   When you click **Add Instance**, a wizard starts and presents the list of available images. Figure 12-8 shows IBM Informix Developer Edition V11.50 as one of the public images available for instantiation. Choose this image, and select the data center where the instance is provisioned from the Select Data Center drop-down menu.



IBM Informix Developer Edition V11.50 - PAYG

IBM Informix Developer Edition V11.50 for Red Hat Enterprise Linux 5.4 (32-bit) with pay as you go use option. There is no-charge for Informix Develo... (Red Hat Enterprise Linux/5.4) [more]

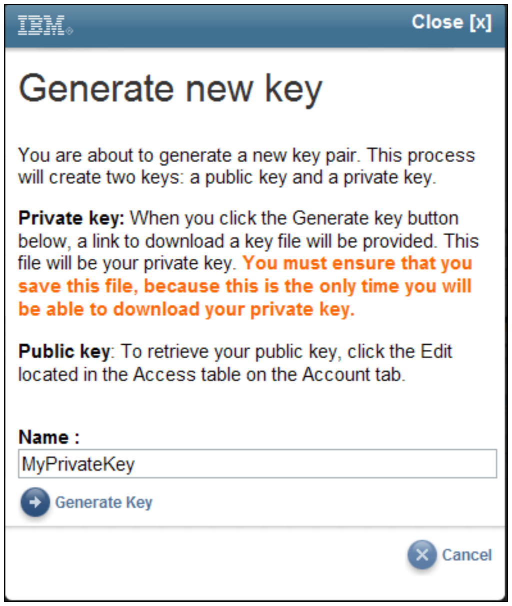
Figure 12-8 Select image

2. View the image details and customize them to your needs.

   Customize the instance by entering information into the configure image window. The following fields must be completed on this page:

   – Request name: The name of the request.
   – Quantity: The quantity of the request.
   – Server size: Server configuration options from the drop-down menu.
   – Minimal local disk: Select to specify that the instance should be provisioned with minimal local storage.
   – Expires on: The expiration date set by the system administrator.
   – Key: The security key for accessing the instance.

Click **Add Key** to generate a new key pair with a public key and a private key. A Generate new key window opens, as shown in Figure 12-9. When you click **Generate key**, a link for downloading the key file is provided. Save the key file in a secure location.



Generate new key

You are about to generate a new key pair. This process will create two keys: a public key and a private key.

Private key: When you click the Generate key button below, a link to download a key file will be provided. This file will be your private key. **You must ensure that you save this file, because this is the only time you will be able to download your private key.**

Public key: To retrieve your public key, click the Edit located in the Access table on the Account tab.

Name : MyPrivateKey

Figure 12-9 Generate new key

– VLAN: If the Public internet option is selected from the drop-down menu, the instance can be accessed over a public Internet-facing VLAN.

– Select IP: The primary IP address for the instance. Select the **system generated** option to use a dynamic IP. If you had created reserved IP address, it also is shown in the drop-down menu.

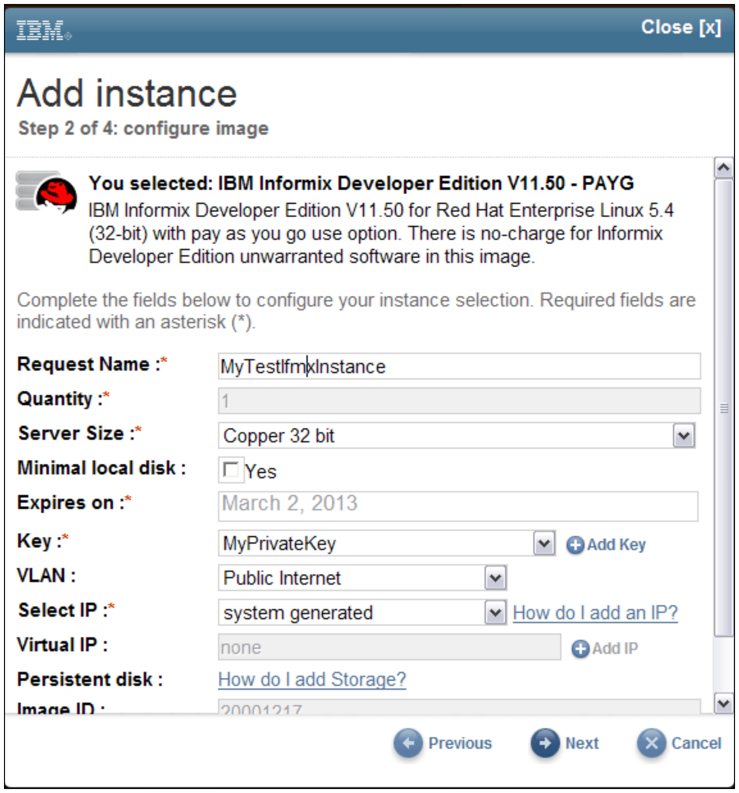
– Virtual IP: Secondary IP address for the instance.

– Persistent disk: Attach a persistent storage disk to the instance. The storage is added through the Storage view of the Control Panel.

– Image ID: The identification of the image.

– Price: The price of the instance based on the image type and application included with the image.

Figure 12-10 shows an example of the information we entered for customizing the Informix instance. After entering the required information, click **Next** to proceed.



Add instance

Step 2 of 4: configure image

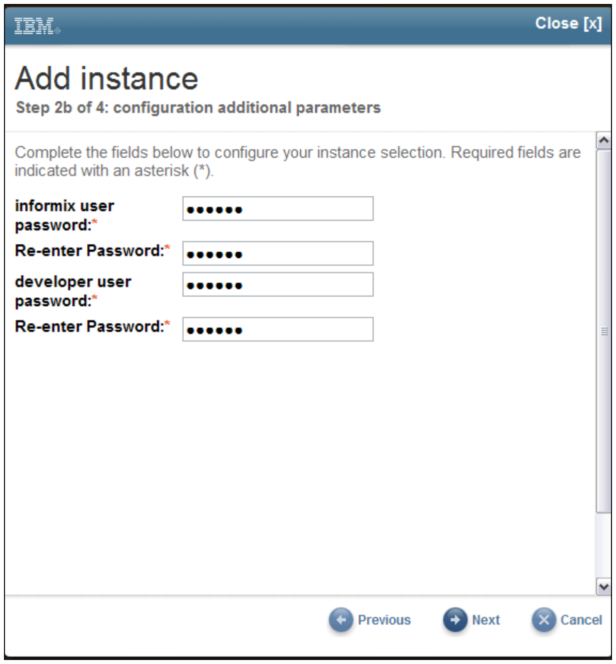
You selected: IBM Informix Developer Edition V11.50 - PAYG
IBM Informix Developer Edition V11.50 for Red Hat Enterprise Linux 5.4 (32-bit) with pay as you go use option. There is no-charge for Informix Developer Edition unwarranted software in this image.

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (*).

Request Name :*	MyTestIfmxInstance
Quantity :*	1
Server Size :*	Copper 32 bit
Minimal local disk :	☐ Yes
Expires on :*	March 2, 2013
Key :*	MyPrivateKey — Add Key
VLAN :	Public Internet
Select IP :*	system generated — How do I add an IP?
Virtual IP :	none — Add IP
Persistent disk :	How do I add Storage?
Image ID :	20001217

Figure 12-10 Configure image

Figure 12-11 shows the next window in the wizard; this window requires you to enter the informix user password and developer user password. After entering passwords for both users, click **Next** to proceed.



Add instance

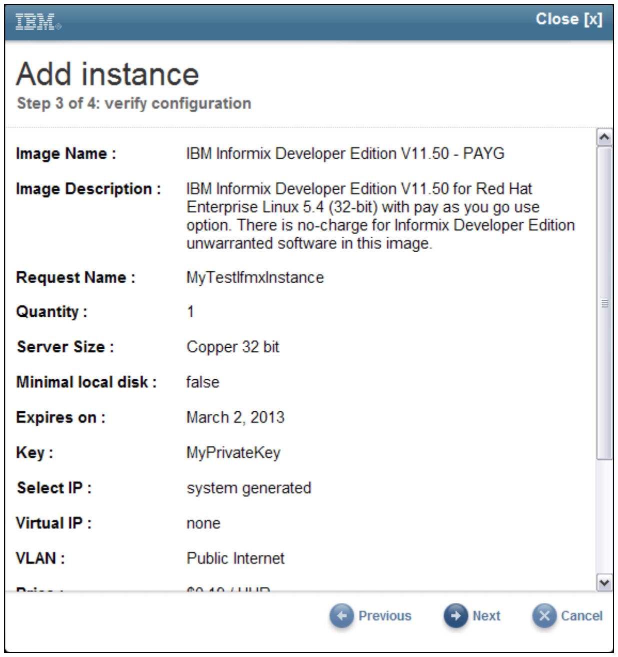
Step 2b of 4: configuration additional parameters

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (*).

informix user password:*	••••••
Re-enter Password:*	••••••
developer user password:*	••••••
Re-enter Password:*	••••••

Figure 12-11 Configuration of additional parameters

Figure 12-12 shows the window that prompts you to verify the image configuration information entered so far. After verifying the configuration, click **Next** to proceed.



Add instance

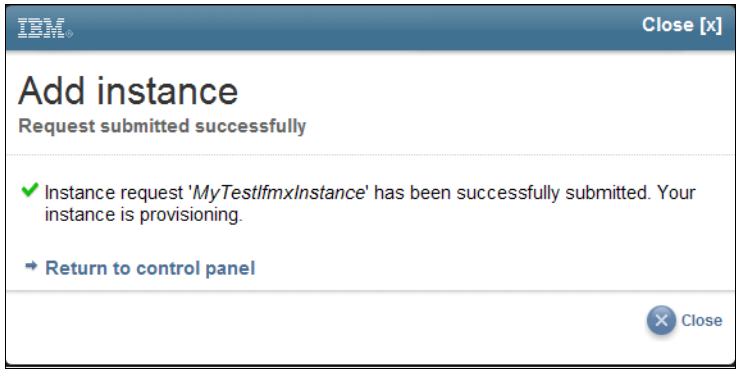
Step 3 of 4: verify configuration

Image Name :	IBM Informix Developer Edition V11.50 - PAYG
Image Description :	IBM Informix Developer Edition V11.50 for Red Hat Enterprise Linux 5.4 (32-bit) with pay as you go use option. There is no-charge for Informix Developer Edition unwarranted software in this image.
Request Name :	MyTestIfmxInstance
Quantity :	1
Server Size :	Copper 32 bit
Minimal local disk :	false
Expires on :	March 2, 2013
Key :	MyPrivateKey
Select IP :	system generated
Virtual IP :	none
VLAN :	Public Internet

Figure 12-12 Verify configuration

The next window displays the service agreement. After agreeing to the service agreement, click **Next** to submit the request for provisioning an Informix instance.

Figure 12-13 indicates that the request was submitted successfully and the instance is currently in the process of provisioning in the IBM Cloud.



Add instance

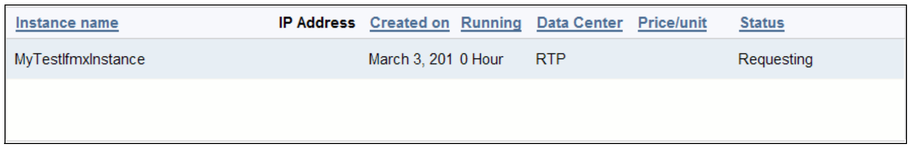
Request submitted successfully

Instance request '*MyTestIfmxInstance*' has been successfully submitted. Your instance is provisioning.

Figure 12-13 Request submitted successfully

3. Watch your instance provision and start managing.

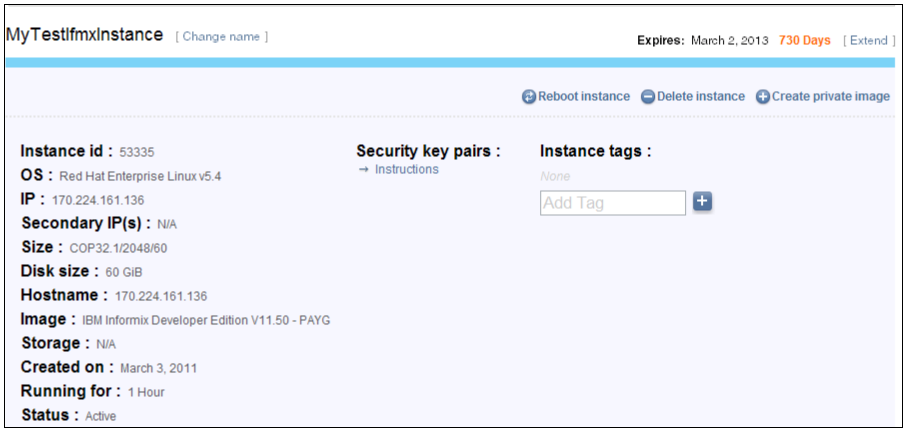
   Figure 12-14 shows the status of the instance provision request inside the Control Panel. The status is shown as "Requesting" immediately after the request is submitted.



Instance name	IP Address	Created on	Running	Data Center	Price/unit	Status
MyTestIfmxInstance		March 3, 201	0 Hour	RTP		Requesting

Figure 12-14 Provisioning status

After few minutes, when the instance is created successfully, the status changes to *Active* and the details about accessing the instance are displayed, as shown in Figure 12-15.


MyTestIfmxInstance [Change name]

Expires: March 2, 2013 730 Days [Extend]

Instance id : 53335
OS : Red Hat Enterprise Linux v5.4
IP : 170.224.161.136
Secondary IP(s) : N/A
Size : COP32.1/2048/60
Disk size : 60 GiB
Hostname : 170.224.161.136
Image : IBM Informix Developer Edition V11.50 - PAYG
Storage : N/A
Created on : March 3, 2011
Running for : 1 Hour
Status : Active

Security key pairs :
→ Instructions

Instance tags :
None

Figure 12-15 Instance in active status

Accessing the instance with PuTTY

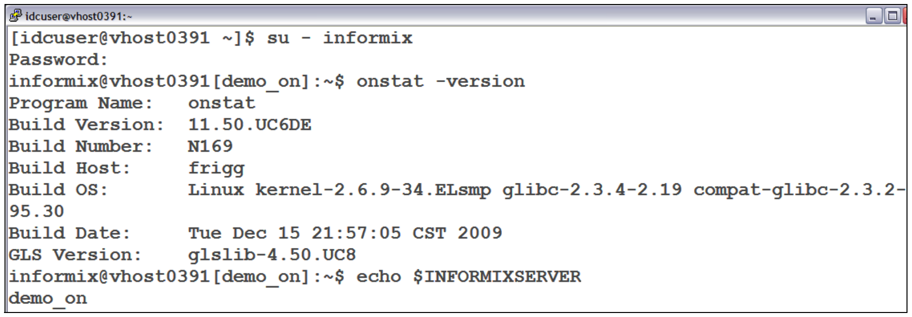
After provisioning an Informix instance successfully, we now show how to connect to the instance using an SSH client on the Microsoft Windows operating system. We chose to use the PuTTY SSH client. The PuTTY and PuTTYgen (for generating key pairs) utilities can be downloaded from the PuTTY download site.

After downloading PuTTY, complete the following steps to connect to the instance:

1. Start PuttyGen to prepare the keys. We obtain the private key associated with the instance from the IBM Cloud portal and convert it to PuTTY format using PuTTYgen. Complete the following steps:
    a. Click **Load private key**, and locate and open the key file (`ibmcloud_<your_name>@us.ibm.com_rsa`) on disk.
    b. Specify a key passphrase and click **Save private key**.
    c. Select the location to save the PuTTY key file and provide the file name, for example, `ibmcloud_<your_name>@us.ibm.com.ppk`, and then click **Save**.
2. Start PuTTY by completing the following steps:
    a. Select **Session** from the category pane on the left, and enter the IP address of the instance displayed inside the Control Panel in the Host Name (or IP address) field.
    b. Select **Connection** → **Data** from the category pane on the left and enter idcuser in the Auto-login user name field.

c. Select **Connection** → **SSH** → **Auth** from the category pane on the left and click **Browse**. Select and open the PuTTY private key file generated before using PuTTYgen.

d. Click **Open** and enter the key passphrase. You are now connected to the instance using SSH. The Informix server demo_on is in the online state, as shown in Figure 12-16.



```
[idcuser@vhost0391 ~]$ su - informix
Password:
informix@vhost0391[demo_on]:~$ onstat -version
Program Name:   onstat
Build Version:  11.50.UC6DE
Build Number:   N169
Build Host:     frigg
Build OS:       Linux kernel-2.6.9-34.ELsmp glibc-2.3.4-2.19 compat-glibc-2.3.2-
95.30
Build Date:     Tue Dec 15 21:57:05 CST 2009
GLS Version:    glslib-4.50.UC8
informix@vhost0391[demo_on]:~$ echo $INFORMIXSERVER
demo_on
```

Figure 12-16 Connected to Informix instance

You are now ready to manage and use the Informix server instance in the IBM Cloud.

12.3.2 Using Informix on the Amazon cloud

We now explain how to deploy Informix on Amazon Elastic Compute Cloud (Amazon EC2), which is a web service providing scalable compute capacity in a public cloud.

The Amazon Machine Images (AMI) for Informix offers a fully functional per-configured database server on Amazon EC2. For our example, we choose the AMI for IBM Informix 11.50 Workgroup Edition on 64-bit, 2.6.16.60 level based SUSE Linux Enterprise Server 10 SP 2 operating system.

Complete the following steps to create an instance based on Informix AMI:

1. Set up an account with Amazon Web Services (AWS) and sign up for Amazon EC2. For details about setting up an account with AWS, go to the following address:

   http://aws.amazon.com

2. Sign in to the Amazon Management Console. Click **Launch Instance** under the AWS EC2 Console Dashboard to start creating an instance.

3. You are directed to a page showing the list of AMIs available for instantiation:
   a. Go to the **Community AMIs** tab and filter the list to show the public images by selecting **Public Images** in the Viewing drop-down menu.
   b. Filter the list further to find the Informix AMIs by searching for "IDS" using the input text box.
   c. Select the AMI for IBM Informix 11.5 Workgroup Edition on 64-bit from the list.
4. You are now presented with a page showing instance configuration details:
   a. Select the **Availability Zone** from the drop-down menu.
   b. Input the Number of Instances.
   c. Select the instance type from the drop-down menu. This action defines the CPU units, CPU cores, and memory for the instance.
   d. You can choose to launch the instance as an *on demand* or *spot* instance. You pay for the capacity by the hour when you choose an on demand instance. Amazon EC2 allows you to bid for unused capacity if you choose a spot instance.
   e. Click **Continue** to proceed after making your selections.
5. In next page, specify the Advanced Instance Options.

   You can select a kernel for the instance by choosing a specific Kernel ID and RAM Disk ID on this page.

   You can also choose **Enable CloudWatch detailed monitoring for this instance** on this page. Amazon CloudWatch service allows you to monitor the instance for resource utilization and performance.

   Click **Continue** to proceed after making your selections.
6. On the next page, you can associate your own metadata for the instance by adding tags to it, which helps you manage the instance. Click **Continue** to proceed.
7. In the page that opens, you can choose a public or private key pair for connecting securely to the instance after launching the instance. Here you can select one of the following options:
   – A key pair that you had created before from the Your Existing Key Pairs drop-down menu.
   – Create a new key pair. You can generate a new key pair from the AWS Management Console by completing the following steps:
      i. Click **Key Pairs** under NETWORKING & SECURITY in the Navigation pane on left.
      ii. Click **Create Key Pair** under the Key Pairs page.

        iii. The **Create Key Pair** window opens and prompts you to enter the new key pair name.
        iv. Enter the new key pair name. Click **Create**.
        v. Download the key file and save it in a safe location.

   – Proceed without a key pair.

   Click **Continue** to proceed after making your selections.

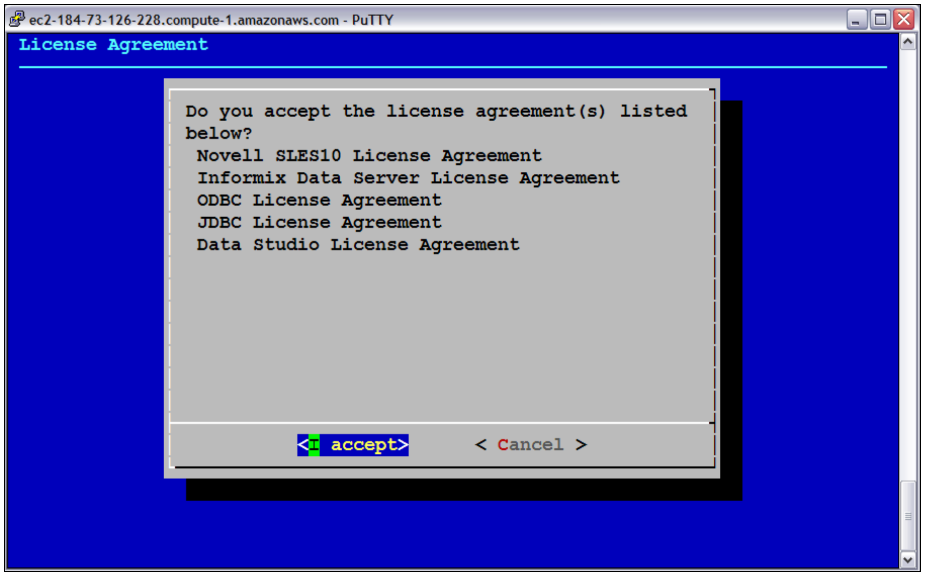
8. The next page displays the Security Groups list. Here you can select one of the following options and specify allowed connections through rules for various connection methods and protocols:
   a. Select the "Default" security group.
   b. A custom defined security group.
   c. Create a new security group.
   d. Click **Continue** to proceed after making your selections.
9. The final page allows you to review the instance configuration information entered so far. Click **Launch** to launch the instance after reviewing the information.
10. AWS displays a page showing that the instance is being launched. Exit the Launch Instance Wizard by clicking **Close**.

Immediately after launching the instance, the status of the instance shows as *pending* on My Instances page. You have to wait for the status to change to *running* before accessing the instance. After the instance is created successfully, its status changes to *running*.

You can look at the details of the newly launched instance by selecting it from the list of instances on My Instances page.

For instructions about accessing the instance, select **Instance Management** → **Connect** from the Instance Actions drop-down menu under the My Instances page. A page with instructions about accessing the instance using a SSH client (such as PuTTY) is presented along with the public DNS of the instance.

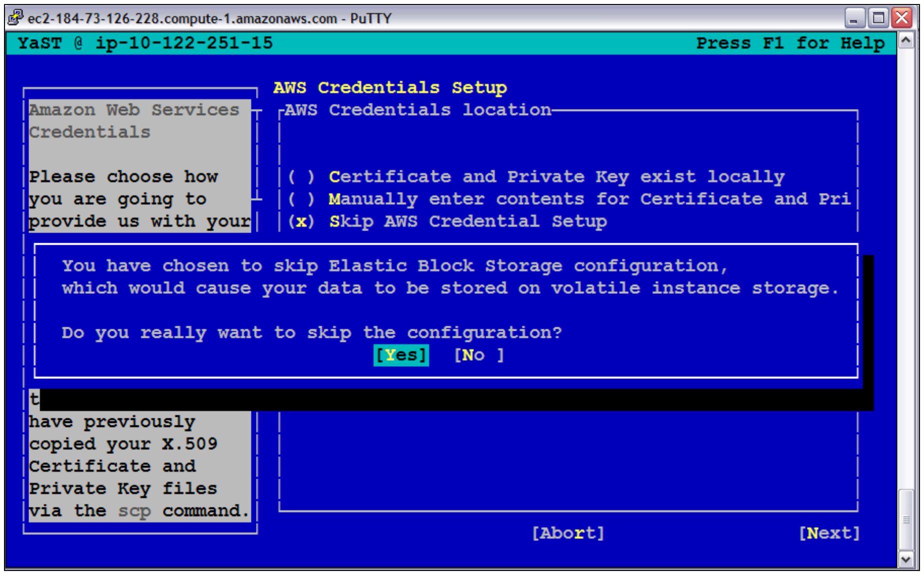
When you access the Informix instance for the first time, a sequence of interactive configuration panels open to help you set up the Informix environment. Figure 12-17 shows the panel that prompts you to accept the license agreement after displaying the license agreement on multiple panels. Accept the license agreement to proceed further.



```
License Agreement

Do you accept the license agreement(s) listed
below?
 Novell SLES10 License Agreement
 Informix Data Server License Agreement
 ODBC License Agreement
 JDBC License Agreement
 Data Studio License Agreement

        <I accept>      < Cancel >
```

Figure 12-17   License agreement

Figure 12-18 shows the panel for setting up AWS credentials for Elastic Block Storage configuration. We choose to skip it and store the data on volatile instance storage in our example.



```
ec2-184-73-126-228.compute-1.amazonaws.com - PuTTY
YaST @ ip-10-122-251-15                                    Press F1 for Help

                      AWS Credentials Setup
Amazon Web Services   AWS Credentials location
Credentials

Please choose how     ( ) Certificate and Private Key exist locally
you are going to      ( ) Manually enter contents for Certificate and Pri
provide us with your  (x) Skip AWS Credential Setup

   You have chosen to skip Elastic Block Storage configuration,
   which would cause your data to be stored on volatile instance storage.

   Do you really want to skip the configuration?
                                [Yes]   [No ]
t
have previously
copied your X.509
Certificate and
Private Key files
via the scp command.
                                          [Abort]              [Next]
```

Figure 12-18   AWS credentials setup

Next, set the root password for the instance, as shown in Figure 12-19. Select **Next** to continue.

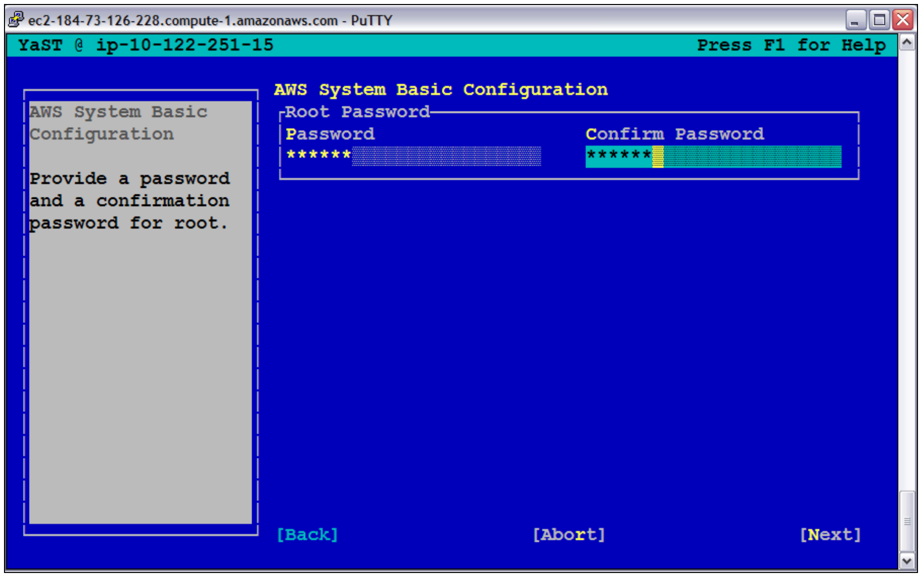
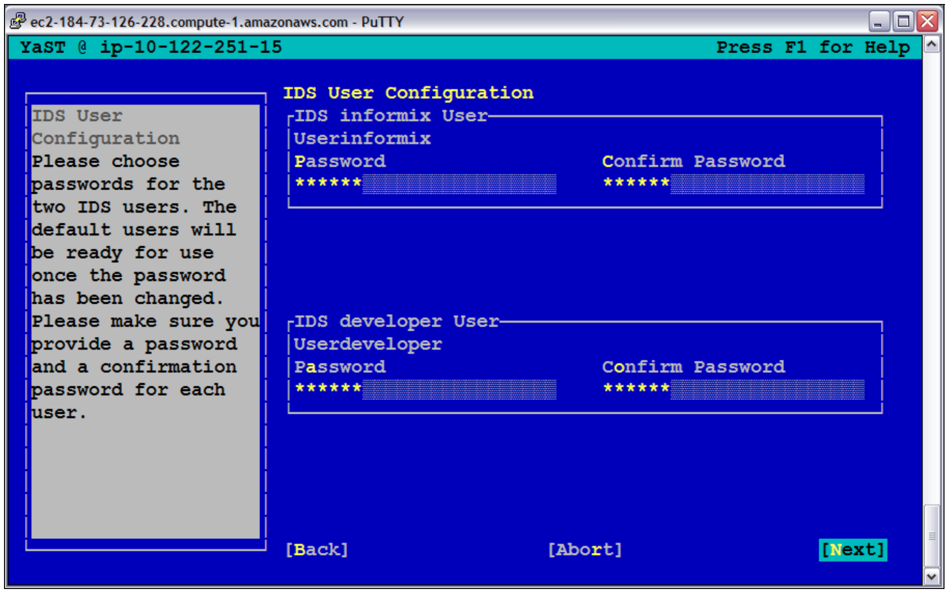

```
ec2-184-73-126-228.compute-1.amazonaws.com - PuTTY
 YaST @ ip-10-122-251-15                                   Press F1 for Help

                      AWS System Basic Configuration
 AWS System Basic     ┌Root Password──────────────────────────────────────┐
 Configuration        │Password                 Confirm Password          │
                      │******                   ******                    │
 Provide a password   └───────────────────────────────────────────────────┘
 and a confirmation
 password for root.



                      [Back]              [Abort]                 [Next]
```

Figure 12-19   AWS system basic configuration

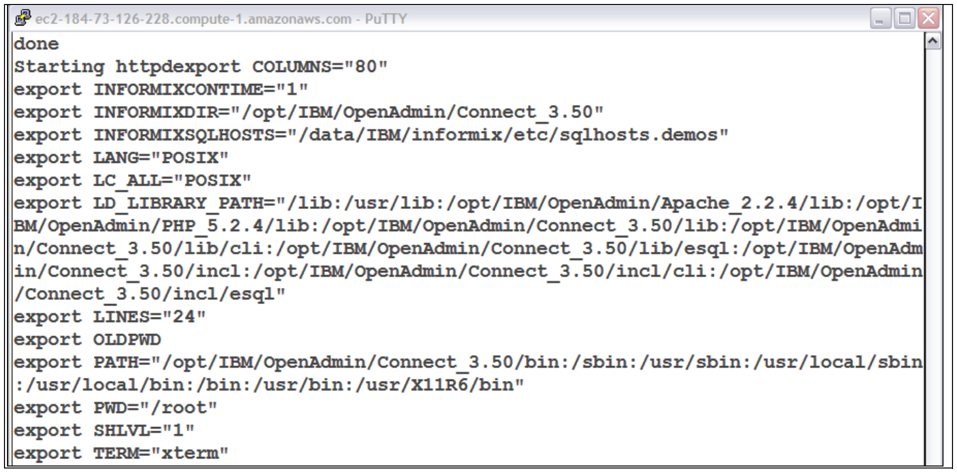
Figure 12-20 shows the panel for setting the Informix user's configuration. Set the passwords for two Informix users: *informix* and *developer*. Select **Next** to continue.



```
YaST @ ip-10-122-251-15                                    Press F1 for Help

IDS User                 IDS User Configuration
Configuration            IDS informix User
Please choose            Userinformix
passwords for the        Password                  Confirm Password
two IDS users. The       ******                    ******
default users will
be ready for use
once the password
has been changed.
Please make sure you     IDS developer User
provide a password       Userdeveloper
and a confirmation       Password                  Confirm Password
password for each        ******                    ******
user.
```

Figure 12-20 IDS User Configuration panel

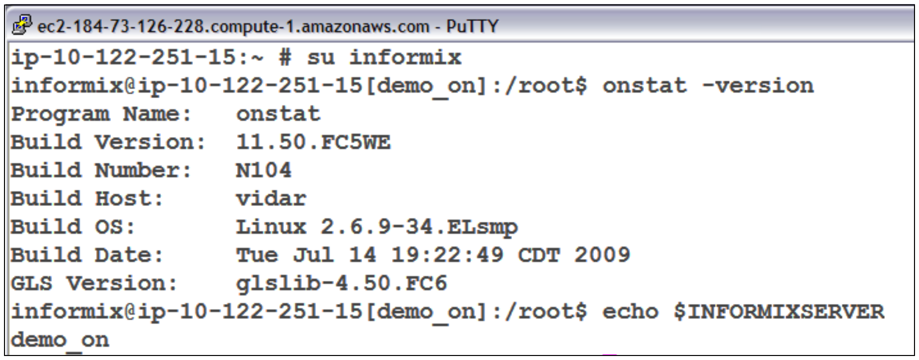
We are now connected to the instance and the environment for the instance is displayed, as shown in Figure 12-21.



```
done
Starting httpdexport COLUMNS="80"
export INFORMIXCONTIME="1"
export INFORMIXDIR="/opt/IBM/OpenAdmin/Connect_3.50"
export INFORMIXSQLHOSTS="/data/IBM/informix/etc/sqlhosts.demos"
export LANG="POSIX"
export LC_ALL="POSIX"
export LD_LIBRARY_PATH="/lib:/usr/lib:/opt/IBM/OpenAdmin/Apache_2.2.4/lib:/opt/I
BM/OpenAdmin/PHP_5.2.4/lib:/opt/IBM/OpenAdmin/Connect_3.50/lib:/opt/IBM/OpenAdmi
n/Connect_3.50/lib/cli:/opt/IBM/OpenAdmin/Connect_3.50/lib/esql:/opt/IBM/OpenAdm
in/Connect_3.50/incl:/opt/IBM/OpenAdmin/Connect_3.50/incl/cli:/opt/IBM/OpenAdmin
/Connect_3.50/incl/esql"
export LINES="24"
export OLDPWD
export PATH="/opt/IBM/OpenAdmin/Connect_3.50/bin:/sbin:/usr/sbin:/usr/local/sbin
:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin"
export PWD="/root"
export SHLVL="1"
export TERM="xterm"
```

Figure 12-21 Environment variables

Figure 12-22 shows that the Informix server *demo_on* is in the online state.



```
ip-10-122-251-15:~ # su informix
informix@ip-10-122-251-15[demo_on]:/root$ onstat -version
Program Name:   onstat
Build Version:  11.50.FC5WE
Build Number:   N104
Build Host:     vidar
Build OS:       Linux 2.6.9-34.ELsmp
Build Date:     Tue Jul 14 19:22:49 CDT 2009
GLS Version:    glslib-4.50.FC6
informix@ip-10-122-251-15[demo_on]:/root$ echo $INFORMIXSERVER
demo_on
```

Figure 12-22 Connect to Informix server

You are now ready to manage and use the Informix server instance in the Amazon EC2 public cloud.

12.4 Cloud usage scenarios

Data center virtualization is becoming a key aspect of IT departments and lays the foundation for cloud environments. Products such as VMware vSphere or VMware Cloud Director are used to manage data centers effectively and for resource allocations. Cloud and virtualization technologies both meet this use criteria: Use a resource for your need (pay for the service in public) and release the resource (no more charge afterwards) when you are done.

Resources are used on demand, and they could be used:

- To provide stand-alone systems: Here the user would need a system for a short period, use it for a quick test (to test performance or a defect that occurs in a particular workload scenario), and then relinquish the use of the resource. Such systems can be in house in virtualized environments or in a public cloud.
- To provide for HA: The database administrator feels that his data environment needs high availability options. Using Informix cloning technology, the DBA can start new nodes either on a cloud or in a virtualization environment to provide these HA options. Virtualization platforms do provide cloning, but Informix cloning makes the database cloning part a lot simpler.

- To add capacity: The database administrator is asked to add capacity for a peak season period. He needs to get a system up and running in a short time. The Informix Flexible Grid capability (along with ifxclone) makes this scenario possible with just a few steps. The Connection Manager setup automatically takes in the new node, and after it is up and running, uses it for load balancing.
- To provide data backup options: Sometimes, vendors want to have external storage that is not on site. Also, they want the database backup and restore process integrated with storage mechanisms. Informix 11.7 provides the ability to back up and restore using the Amazon cloud S3 storage offering.
- To use the capabilities provided by virtualization platforms: VMware provides a technology called VMotion, which can be used in scale-up scenarios (the need to move your virtual machine to a host with higher capacity). In such cases, the databases are expected to work without issues post migration. The configuration before and after should not change and things like Connection Manager should work without issues. Naturally, parameters should be configured at the database server or related components to make this migration work effectively. For example, the Connection Manager failover control (FOC) arbitrator should not kick in when the migration is in progress, so the time should be setup appropriately.

We describe some of these models with examples about how you would set them up.

12.4.1 Stand-alone systems

A simple stand-alone system for a quick test setup or a topology based system (such as a cluster or grid) can be set up on both the public and private cloud environments. We saw examples about how simple stand-alone systems can be set up on both the IBM Cloud and the Amazon cloud. Also, Amazon lets you bring in your own Informix license in a model called *bring your own license* (BYOL).

In this model, you provision infrastructure on the Amazon cloud and then install your Informix instance on it. You can provision any number of instances using your own licenses for setting up stand-alone or topology systems.

Stand-alone and topology systems can also be set up in the private clouds. As mentioned in Chapter 1, “Overview” on page 1, private clouds are on site. Informix installs capabilities such as Deployment Assistant, Deployment Utility, and ifxclone that can help you set up these private clouds with ease.

12.4.2 Expanding capacity (VMware and no VMware environments)

With Informix 11.70, you can easily add a new server to a grid for expanding capacity by cloning an existing replication server in the grid. In a virtualized environment, you can add virtual machines to host the clone server on demand.

In this section, we present an example where we create a grid with two servers on two virtual machines in a VMWare environment, to which we add a third server, also on a virtual machine. Here are the steps we perform to achieve this task:

1. Install Informix 11.70.UC1 on VM1 (cisqa18vm1) and VM2 (cisqa18vm2).
2. Start Informix server instances on both virtual machines (server ol_informix1170_1 on VM cisqa18vm1 and server ol_informix1170_2 on VM cisqa18vm2). Create a smart blob space named sbspace on both severs.
3. Create the `sqlhosts` file on both the virtual machines with ER groups, as shown in Example 12-2.

Example 12-2 sqlhosts file

```
g_ol_informix1170_1 group - - i=4930
ol_informix1170_1 onsoctcp cisqa18vm1 ol_informix1170_1
g=g_ol_informix1170_1

g_ol_informix1170_2 group - - i=4931
ol_informix1170_2 onsoctcp cisqa18vm2 ol_informix1170_2
g=g_ol_informix1170_2
```

4. Using OpenAdmin Tool (OAT), create an ER domain as described in Chapter 3, “Configuring a server for Flexible Grid and Enterprise Replication environments” on page 31. We select the ER node type as *root node* for both the servers.
5. Create a grid with two servers, as described in Chapter 4, “IBM Informix Flexible Grid” on page 57. We select server ol_informix1170_1 as the source server.
6. We then proceed to add a new server to the grid by cloning the existing replication server ol_informix1170_1 using the ifxclone utility on the third virtual machine VM3 (cisqa19vm1). The virtual machine has the same operating system as the source server (ol_informix1170_1**)** we are cloning. Complete the following steps:
   a. Set the value of the ENABLE_SNAPSHOT_CLONE parameter in the `onconfig` file for ol_informix1170_1 to 1.
   b. Install Informix 11.70.UC1 on VM3.

c. On VM3, we set the environment variables and required configuration parameters in the `onconfig` file for the new server instance (ol_informix1170_3) to satisfy the prerequisites for ifxclone. We must set the required configuration parameters to the same values as on the ol_informix1170_1 server.

d. On VM3, create all the chunks that exist on the ol_informix1170_1 server.

e. Update the `sqlhosts` file on each server with information for all the servers in the grid.

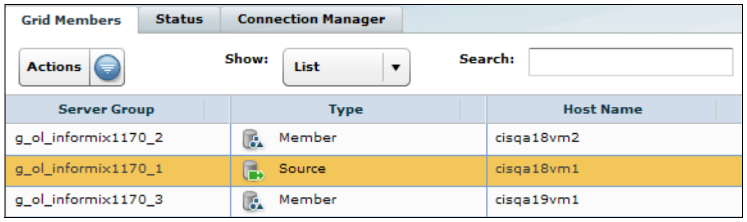
f. Run the ifxclone utility to create a clone of the ol_informix1170_1 server with a final disposition of ER, as shown in Example 12-3.

Example 12-3 Clone a server on another VM

```
ifxclone -S ol_informix1170_2 -I cisqa18vm2 -P ol_informix1170_2 -t
ol_informix1170_3 -i cisqa19vm1 -p ol_informix1170_3 -L -T -d ER
```

g. The ol_informix1170_3 server is automatically added to the grid.

Figure 12-23 shows a view of this grid in OAT created across three servers on the virtual machines.



Server Group	Type	Host Name
g_ol_informix1170_2	Member	cisqa18vm2
g_ol_informix1170_1	Source	cisqa18vm1
g_ol_informix1170_3	Member	cisqa19vm1

Figure 12-23 Grid members

Example 12-4 shows the output of the **`cdr list grid`** command for the same grid.

Example 12-4 cdr list grid

```
cdr list grid

#output
Grid                     Node                     User
------------------------ ------------------------ ------------------------
grid_ol_informix1170     g_ol_informix1170_1*     informix
                         g_ol_informix1170_2
                         g_ol_informix1170_3
```

12.4.3 Cloud storage for database backup(s)

Informix 11.70 allows its data to be backed up to a cloud storage location at Amazon Simple Storage Service, also called Amazon S3, using the ontape backup and restore utility.

Using a cloud storage location for database backups offers many advantages, and can supplement an organization's strategy to back up data at an off-site location. The backup data stored at Amazon S3 can be accessed from anywhere on the internet. There is no requirement to plan for and set up a storage infrastructure, and the storage can grow with backup data as needed.

Though this integrated approach makes it convenient to automate Informix backups to an off-site storage, the user is responsible for the terms and any charges associated with the use of Amazon S3.

The user must set up an account with Amazon S3 before performing backups to Amazon S3. The Amazon Web Services (AWS) website has instructions about setting up an account.

Further, Java Version 1.5 or later is required in the environment from where ontape will perform backup and restore operations.

Here are the steps required before backing up Informix data to Amazon S3. The size of level 0, 1, and 2 backups or each logical-log backup is limited to five GB in Informix 11.70.xC1.

1. After signing up with Amazon S3, obtain the access credentials (access key ID and the secret access key) associated with the account, and store them in a file at `$INFORMIXDIR/etc/ifxbkpcloud.credentials` on UNIX (or in a file at `%INFORMIXDIR%\etc\ifxbkpcloud.credentials` on Windows). Example 12-5 shows the format of the access credentials file.

   Example 12-5 Access credentials file

   ```
   secretKey=<secret_access_key>
   accessKey=<access_key_ID>
   ```

2. For security reasons, set the permissions on the file such that only the user executing **`ontape`** has access to it.
3. Create the cloud storage device (or bucket) in the region where you plan to store backup data. You may use the ifxbkpcloud.jar utility distributed with the Informix product to create the device. Amazon S3 requires the name of the storage device to be unique.

Example 12-6 shows how to create a storage device named ibm-ifx-amazons3 in US Standard region.

Example 12-6 Create storage device

```
$INFORMIXDIR/bin/java -jar ifxbkpcloud.jar CREATE_DEVICE amazon
ibm-ifx-amazons3 US_Standard
```

4. Specify cloud storage as a backup storage device by setting the TAPEDEV and LTAPEDEV parameters in the `onconfig` file.

   Example 12-7 shows how to set the TAPEDEV configuration parameters in the `onconfig` file. The fields have the following meanings:

   – The first field is set to the path name of the directory where storage spaces backup objects are stored temporarily.

   – The second field is set to yes to specify that the ontape utility retains the backup objects in the local directory.

   – The third field is set to the name of the cloud storage vendor (Amazon in this case).

   – The fourth field is set to the URL of the cloud storage location.

   Example 12-7 Tape device configuration parameter

   ```
   TAPEDEV '/opt/IBM/informix/tapedev_dir, keep = yes, cloud = amazon,
   url = https://ibm-ifx-amazons3.s3.amazonaws.com'
   ```

   – You can set the LTAPEDEV configuration parameter in a similar manner.

After configuring the backup storage location and **`onconfig`** parameters, you are ready to perform backup and restore operations with ontape to Amazon S3 cloud storage.

Example 12-8 shows a level 0 backup taken to the cloud storage location configured above. The backup object stored at the cloud location has the same name as the file created in the local directory for temporary store.

Example 12-8 Backup to Amazon S3 cloud storage

```
informix@informixva[demo_on]:/opt/IBM/informix$ ontape -s -L 0
100 percent done.
Backing up file /opt/IBM/informix/tapedev_dir/informixva_0_L0 to cloud storage
location https://ibm-ifx-amazons3.s3.amazonaws.com.
File created: /opt/IBM/informix/tapedev_dir/informixva_0_L0

Please label this tape as number 1 in the arc tape sequence.
This tape contains the following logical logs:
```

```
1

Program over.
```

Example 12-9 shows a part of the output during a cold restore from the same cloud storage location. First, the backup object stored at the cloud location is retrieved to the local directory. After that, the server is restored from the local backup object.

Example 12-9 Restore from a cloud storage

```
informix@informixva[demo_on]:/opt/IBM/informix$ ontape -r
Restoring file /opt/IBM/informix/tapedev_dir/informixva_0_L0 from cloud storage
location https://ibm-ifx-amazons3.s3.amazonaws.com. Restore will use level 0
archive file /opt/IBM/informix/tapedev_dir/informixva_0_L0. Press Return to
continue ...
```

To keep your data secure, use https secure data transmission for transferring data to and from Amazon S3, and use the BACKUP_FILTER configuration parameter, which calls an external encryption program to encrypt the data before sending it to Amazon S3. The encrypted data can later be restored by setting the RESTORE_FILTER configuration parameter to a corresponding external decryption program.

If the object already exists at the cloud storage location, the file is renamed to avoid overwriting the old object.

12.4.4 Cloud for high availability

Let us look at a scenario in which we have a standard Informix server configured to run an OLTP workload, and a high availability solution is being considered to provide protection from site failure.

We plan to add a Remote Secondary Standby (RSS) server to the current configuration, which is geographically distant from the site. Unable to find a suitable machine for hosting RSS at our sites, we decided to run the RSS server in a public cloud, such as IBM Smart Business Development and Test on the IBM Cloud.

We first provision a virtual server instance in the cloud with a similar operating system and configuration as the in-house server configured to run as the primary server in the high availability setup. After provisioning, we install the Informix product with the same version as the primary server version on the virtual server instance.

Alternatively, if we provision an instance based on an Informix image, it already has the Informix product installed on the instance and a pre-configured server started when the virtual server instance was created. In that case, we must stop the running Informix server and set up its configuration for RSS.

Here are the approaches we can implement to set up RSS:

- Start RSS on the virtual server instance in the cloud using the ifxclone utility by completing the following steps:
  a. Set the environment variables and required configuration parameters in the `onconfig` file on the instance in the cloud to satisfy the prerequisites for ifxclone. We need to set the required configuration parameters to the same values as on the Primary server**.**
  b. After setting up the configuration on the cloud instance, run the ifxclone utility with the disposition set to RSS.
- If you are using Amazon Web Services (AWS), complete the following steps:
  a. Back up the primary server to a cloud storage location in Amazon S3, as shown in 12.4.3, "Cloud storage for database backup(s)" on page 490.
  b. Restore the secondary server from this backup in the cloud to an Informix instance in the Amazon EC2 cloud.

  This strategy can also serve the purpose of storing backups of the primary server in Amazon S3 cloud storage.

  You can also use this approach to set up an HDR server for high availability.

12.4.5 A migration scenario with VMware VMotion

The VMware VMotion can be used for scale up scenarios or can be used by the VMware cluster technology to move a virtual machine automatically in case of any failure. It is expected that the Informix setup is not affected by migrations and continues to function the way it was before the changes. A good understanding about how to set up the VMware vCluster environment is assumed. Also, care should be taken with setting parameters on the Informix server and related components to appropriate values so that they do not conflict with the VMware VMotion part. An example is the Connection Manager failover arbitrator TIMEOUT setting.

In this migration scenario, we demonstrate the movement of an Informix server from one machine to another (a less powerful to a more powerful one). As the server is being moved, the server and the components (for example, Connection Manager) that are working with the server are able to adapt to the transition. Connections are routed to other nodes in the grid when the server being moved is not available, and as soon as the server becomes available, it becomes part of the Connection Manager's pool of resources.

We have a cluster defined with two machines running VMware ESXi virtualization technology and each machine runs a Red Hat virtual machine with Informix 11.7 installed on it. The Informix setup is a simple grid with two nodes and one Informix Connection Manager responsible for the load balancing between them. The Connection Manager is run in a third virtualized environment (not part of the cluster).The VMotion is run using vSphere Client.

Complete the following steps to test the migration:

1. Configure the servers on the two Red Hat virtual machines and start them using **`oninit -ivy`**. Example 12-10 shows the `sqlhosts` file for the two servers.

   Example 12-10 Sqlhosts file for the two servers (identical in both machines)

   ```
   gr_1    group    -       -       i=2000
   ol_informix1170 onsoctcp cxv20000 ol_informix1170 g=gr_1
   gr_2    group    -       -       i=2001
   ol_informix1170_2 onsoctcp cxv20001 ol_informix1170_2 g=gr_2
   ```

2. Create an ER domain using OAT. Add the two servers to a single ER domain.
3. Define a grid called "esxgrid" with two nodes using OAT. Make one of the nodes the source server. You can check the grid's status by running **`cdr list grid`**. Example 12-11 shows the output.

   Example 12-11 cdr list grid command run on the source server

   ```
   Grid                     Node                     User
   -------------------- --------------------   ------------------------
   esxgrid               gr_1*                  informix
                         gr_2
   ```

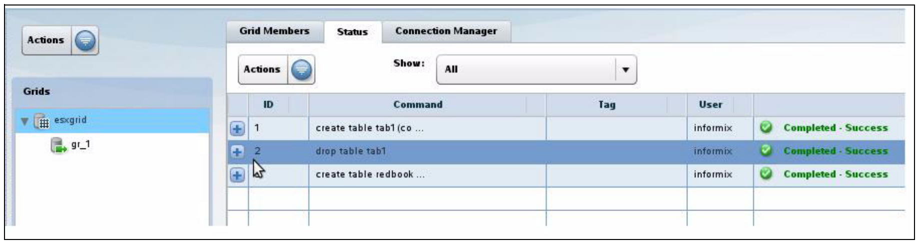
4. Make sure the grid is active by running CREATE TABLE statements with some inserts and verify that the commands are replicated using OAT. Example 12-12 show the SQL script for verifying the grid setup

   Example 12-12 SQL script

   ```
   execute procedure ifx_grid_connect("esxgrid",1);
   create table redbooktab(stuid serial,name char(100));
   ```

```
insert into redbooktab(name) values("Venkatesh G");
insert into redbooktab(name) values("Whei-Jen C");
insert into redbooktab(name) values("Madison P");
insert into redbooktab(name) values("Holgar k");
insert into redbooktab(name) values("Ammu S");
insert into redbooktab(name) values("Amit V");
execute procedure ifx_grid_disconnect();
```

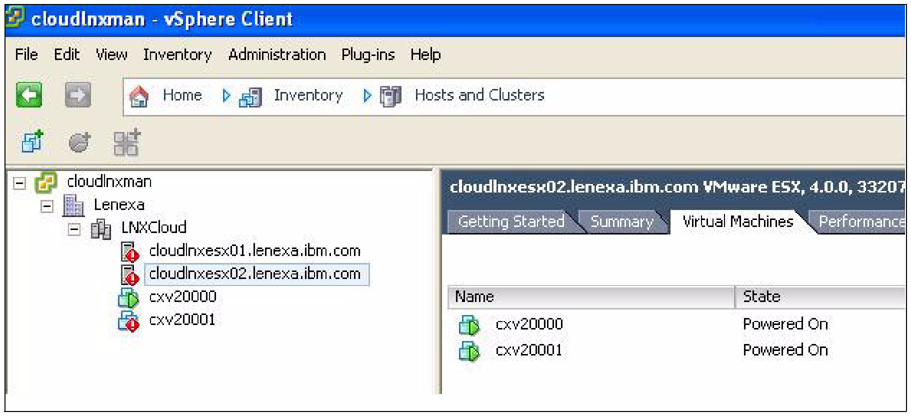
Figure 12-24 shows the OAT window showing that the SQL statements have been propagated from the primary server to the other server.



ID	Command	Tag	User	
1	create table tab1 (co ...		informix	Completed - Success
2	drop table tab1		informix	Completed - Success
3	create table redbook ...		informix	Completed - Success

Figure 12-24 The grid commands are being propagated

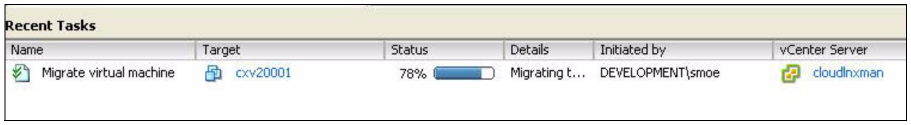
5. Run Connection Manager on a third machine to balance workload across the two nodes in the grid. Ensure that the Connection Manager is registered using **`onstat -g cmsm`** at one of the nodes.



Name	State
cxv20000	Powered On
cxv20001	Powered On

Figure 12-25 Virtual machines located in a single ESX server

6. Try the migration now. Figure 12-25 on page 495 shows server (Virtual machine) locations before the migration. Both of the virtual machines (where the servers are installed) are in a single host (cloudlnxesx02). The migration is done using VMware VMotion technology. VMware VMotion technology lets you move the virtual machine to another host that is part of the cluster. This action is done by right-clicking a particular host and then moving it to a different host. In this example, we move cxv20001 to a different host. Figure 12-26 shows that the migration is in process and that the cxv20001 machine is being moved.

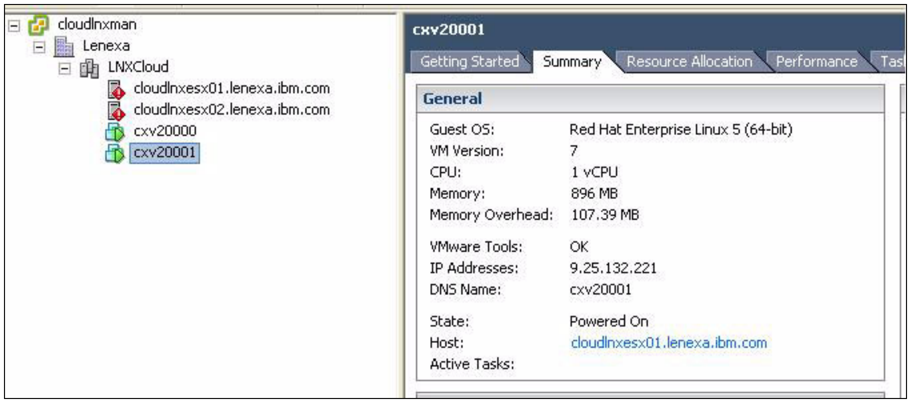


Recent Tasks

Name	Target	Status	Details	Initiated by	vCenter Server
Migrate virtual machine	cxv20001	78%	Migrating t...	DEVELOPMENT\smoe	cloudlnxman

Figure 12-26 Server location before migration

7. Look at where cxv20001 resides now. It has been moved to a different host, as shown in Figure 12-27.



cxv20001

General

Guest OS:	Red Hat Enterprise Linux 5 (64-bit)
VM Version:	7
CPU:	1 vCPU
Memory:	896 MB
Memory Overhead:	107.39 MB
VMware Tools:	OK
IP Addresses:	9.25.132.221
DNS Name:	cxv20001
State:	Powered On
Host:	cloudlnxesx01.lenexa.ibm.com
Active Tasks:	

Figure 12-27 Location of the virtual machine after migration

8. Look at the servers, grid, and the Connection Manager's status after migration. There are no changes at the server side. Note that in this example that the transition was fast. However, it could happen that the Connection Manager might see that the node is not available and will route applications to other nodes that are part of the grid. As soon as the node is ready, Connection Manager adds it back to its list of the available nodes in the grid.

9. Run some commands with the grid and note that they are propagated (Example 12-13).

Example 12-13 Script to verify the environment after migration

```
execute procedure ifx_grid_connect("esxgrid",1);
insert into redbooktab(name) values("I have moved ");
insert into redbooktab(name) values("to a new location");
execute procedure ifx_grid_disconnect();
```

A similar exercise with Informix has been done on HP IntegrityVM using Mach11 clusters and is documented at the following address:

http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA0-3857ENW.pdf

12.5 Administering Informix in a cloud environment

You can administer an Informix server in the cloud in the same way as you administer it outside of a cloud. Use either OpenAdmin Tool (OAT) or the command-line tools, such as onstat and onmode, to manage the server.

To access the Informix server through OAT, you must open the port number for the server.

12.6 Informix Flexible Grid and cloud environments

Informix is a good fit for cloud environments, whether they are public or private. There are several features in the Informix server that are useful for cloud deployments, such as autonomics, embeddability, security, ease of deployment, high availability, and so on. All of these features are useful in different cloud scenarios.

Another such feature is Flexible Grid. In cloud or virtualization environments, you are not guaranteed the same architecture every time. The architecture depends on how the IT department or the IT environment of the cloud provider is set up. Most of the virtualization is created on an existing infrastructure and it may not provide homogeneous work environments. A solution that can work with a heterogeneous system is required for scale-out scenarios. New nodes need to be provisioned with ease and the node has to immediately participate and handle workloads. The Flexible Grid ifxclone utility provides a quick setup (from a source node) and then the Connection Manager automatically makes the node participate in the workload.

13

Migration

Database server migration can be categorized into two major types: in-place migration and non-in-place migration. For an in-place migration, you do not have to use data migration tools or utilities. The data is converted automatically from the source database server to the target database server after you start the target database server. For non-in-place migration, you might need to move data to a new architecture or a different server. This type of migration is more complicated than in-place migration. It requires more planning and it is more time consuming.

In this chapter, we describe how to perform in-place migration (also known as an *upgrade*) of the Informix database servers participating in a replication environment. First we show you how to upgrade the database servers and applications in an Enterprise Replication and Flexible Grid environment. Then we show you how to upgrade database servers that are part of high availability cluster and Connection Manager. The idea here is to demonstrate how upgrades can be performed with minimal impact to the database users and applications during the entire upgrade process.

13.1 Upgrade considerations in a replication environment

Both Flexible Grid and Enterprise Replication supports data replication across different versions of Informix database server, which gives you the ability to upgrade one node at a time and the flexibility in planning an upgrade of all the database servers involved in replication over a period of time rather than upgrading everything at once. This process is known as a *rolling upgrade*. You can upgrade database servers and database schema along with applications in this manner with minimal downtime.

13.1.1 Upgrading replication servers

To upgrade replication servers one at a time, you have to be able to isolate the database server being upgraded from the replication environment and from the database users and applications.

Figure 13-1 shows the typical replication environment where two replication servers, newyork and london, use Informix 11.50 for data replication, and Connection Manager is configured to balance the application workload among those two replication servers. Additionally the connection manager SLA specify an attribute *POLICY=FAILURE* that redirects clients to the london server if the newyork server is down.

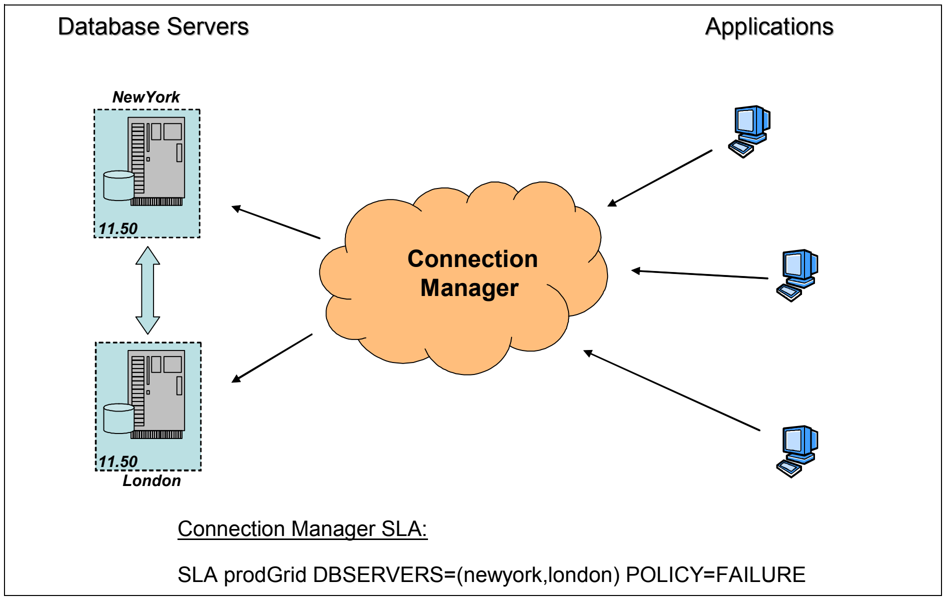


Figure 13-1 Replication environment before an upgrade

The typical procedure to upgrade an replication server is:

- Ensure that automatic application redirection is set, so that when you shut down the server, it will not affect applications.
- The server that is going to capture all the transactions during an upgrade should have enough CDR_QDATA_SBSPACE configured for storing spooled row data.
- On the server that you plan to upgrade:
  - Ensure that all the data replication queues are empty by running **`onstat -g rqm`**.
  - Shut down Enterprise Replication using **`cdr stop`**.
  - Upgrade the server to a new version.
  - Convert the syscdr database to a new version.
  - Start ER on the newly converted server.

To convert a syscdr database, use the concdr.sh script under the `$INFORMIXDIR/etc/conv` directory on UNIX, or concdr.bat under the `%INFORMIXDIR%\etc\conv` directory on Windows.

For detailed steps, go to the following address:

http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.mig.doc/ids_mig_054.htm

Figure 13-2 shows the configuration after you stop the replication at the newyork server.

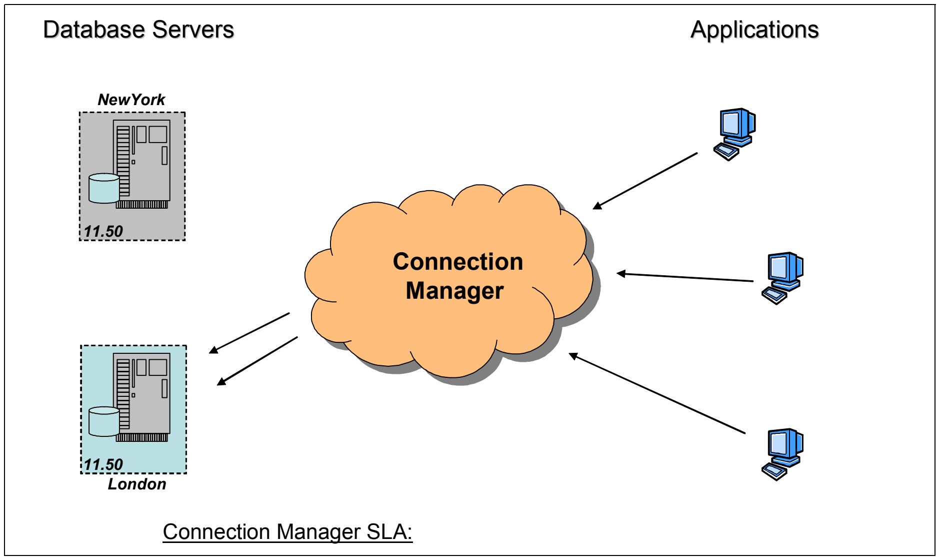


Figure 13-2 Replication environment during an upgrade

The newyork server is now isolated from the database users and applications, but users and applications continue to work with the london replication server while you perform an upgrade of the newyork server.

After the upgrade is complete and you restart the replication at the newyork server, the replication environment appears as shown in Figure 13-3.

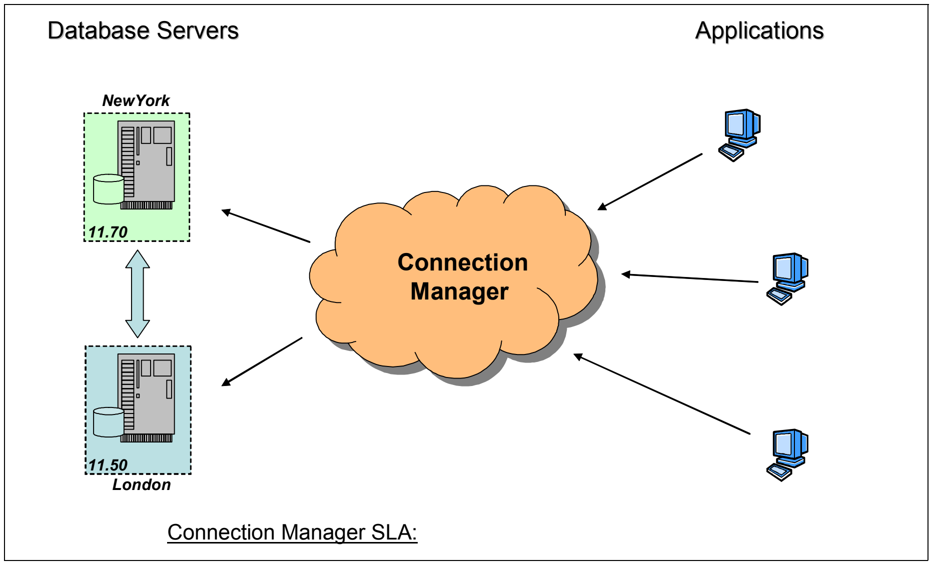


Figure 13-3   Replication environment after an upgrade

As the newyork server becomes available, the replication server, london, sends all the updates that was captured while the newyork server was undergoing an upgrade. At the same time, Connection Manager also start redirecting applications to the newyork server as well.

Note that after an upgrade, the replication server london, which is using Informix 11.50, is replicating data changes to and from the newyork replication server, which is using Informix 11.70. This is the biggest advantage of Flexible Grid and Enterprise Replication.

Repeat the process to upgrade other database servers in the replication environment.

Note: For more information about upgrading the server, refer to the *IBM Informix Migration Guide* found at the following address:

```
http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.mig.doc/
mig.htm
```

13.1.2 Upgrading database schema and applications

A rolling upgrade can be used to upgrade the schema of the replicated table along with the deployment of a new application that works against the newer version of the table schema. A rolling upgrade of applications and database schema is not necessary if applications are written in such a way that even if the underlying table schema changes, applications continue to work with the new table. However, most of the database applications are tightly coupled to the table schema, so a rolling upgrade of an application is needed.

To change the schema of the replicated tables dynamically while replication is active, replicates must be defined as the mastered replicate. By default, when you create replicates using a replication template or Flexible Grid, they are defined as master replicates. If a replicate is not defined as a master replicate, then you could update the replicate to be a master replicate using **`cdr remaster`**.

As an example, Figure 13-4 shows how two replication servers, newyork and london, are replicating a database table, named Tablex, which has four columns. A set of applications, applications V1, are connected to each of the database servers and are executing transactions against Tablex. Connection Manager is used for application redirection and workload balancing.

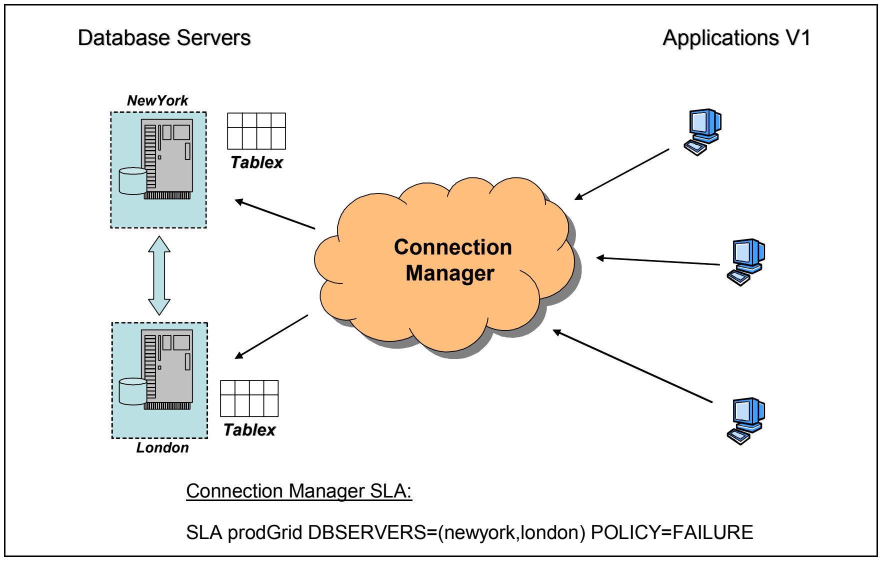


Figure 13-4 Replicated table with applications executing

Now, let us assume that you want to add two columns to Tablex on both the servers and deploy the new version of the application (Applications V2), which is also developed to access the new version of the table Tablex.

To upgrade the table schema and applications in this environment at the newyork server, first redirect all the applications, Applications V1, running against the newyork server to the london server. Redirection of applications to the london server can be done by using Connection Manager. Update the Connection Manager SLA, as shown in Figure 13-5, to remove the newyork server from the list of available database servers for applications V1 and reload the Connection Manager configuration.

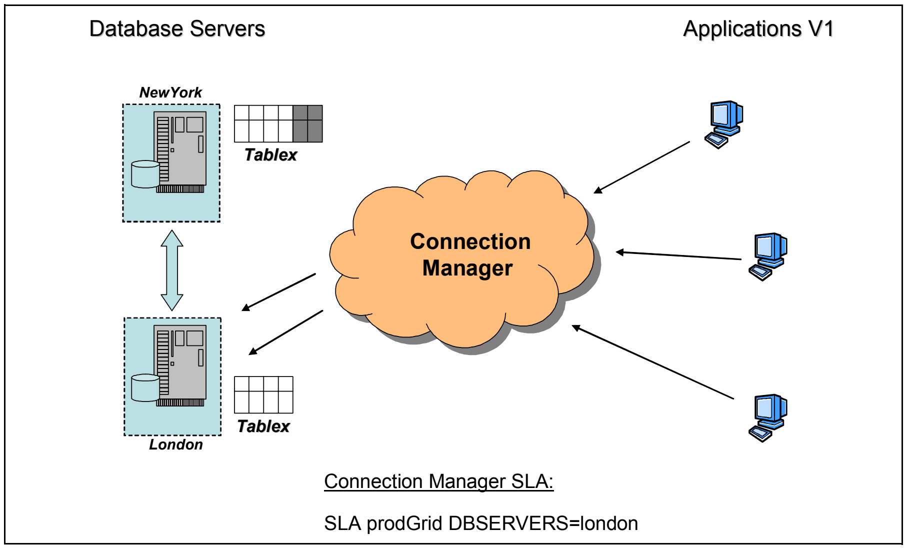


Figure 13-5 Redirected applications and altered table schema

Then change the Tablex schema, using the SQL ALTER TABLE statement, and add two columns to it. Data replication will be active and replicate data changes done at the london server to the existing four columns of the Tablex table.

Now that the newyork database server has a new table schema, you can retire the old version of the application Applications V1 and deploy the new application, Applications V2, at both sites, but all the new applications will now be connecting to the newyork server, as shown in Figure 13-6. Note that to redirect all applications to newyork, you have to update the Connection Manager SLA to remove the london server and add the newyork server. An updated SLA is shown in Figure 13-6.

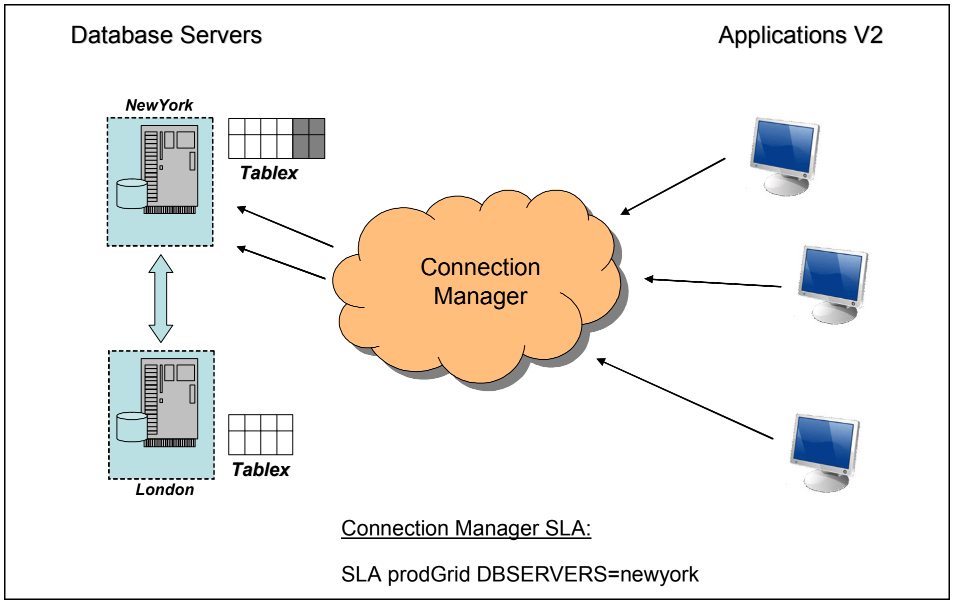


Figure 13-6 Deployment of Applications V2

At the london replication server, the Tablex table can now be altered to add those two extra columns.

After the Tablex schema has been updated on both database servers, you have to remaster the replicate definition, using **`cdr remaster`**, to include the new columns in the replicate definition. After a replicate is remastered, data changes done on the newly added columns start replicating to the other server.

Finally, you can load balance the application workload among both replication servers, as shown in Figure 13-7. Thus, without any server downtime, we can smoothly upgrade the applications from one version to another version.

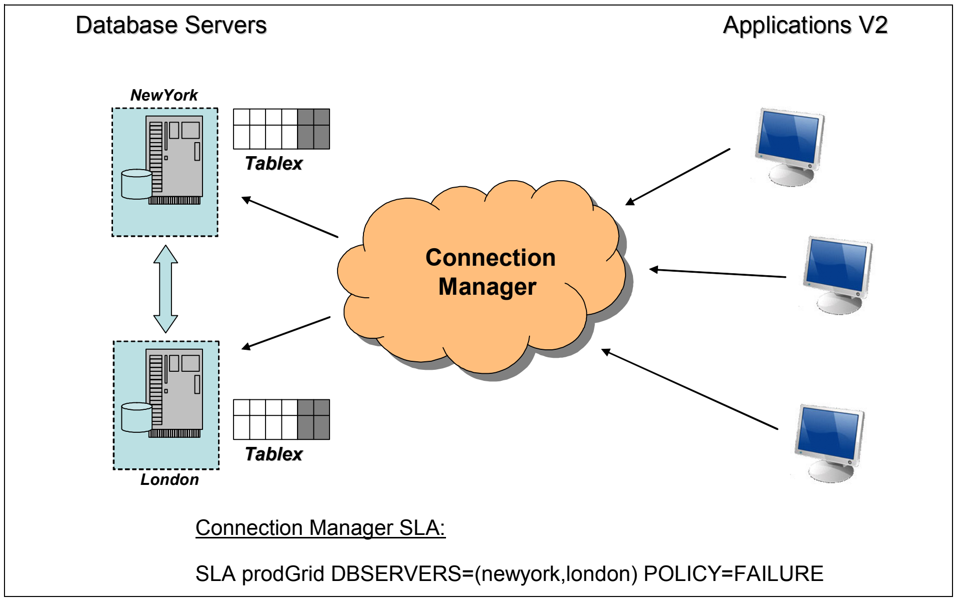


Figure 13-7 Configuration after a rolling upgrade of the application and table schema

13.2 Upgrading a MACH11 cluster environment

Migration can be handled easily by the Informix database software. There are various options available to upgrade, such as upgrading to a fix pack or a post interim drop (PID), upgrading to a new release of IBM Informix or upgrading a cluster online, which is a new functionality available with Version 11.70. Database server administrators (DBAs) who are responsible for upgrading the database server can pick one of the available choices based on their business needs.

In this section, we describe how to:

- Prepare a cluster for an upgrade.

  There are certain preparations you must make to have a successful upgrade.
- Upgrade a cluster to a new fix pack or PID.

  When moving to a new fix pack or PID, you do not have to recreate the secondary servers after you upgrade the primary server.

- Upgrade a cluster to a new release.

  When moving to a new release prior to Informix 11.70, you must shut down the cluster and upgrade only the primary server. After upgrading the primary server, you must recreate all the secondary servers in the high-availability cluster.

- Upgrade a cluster while it is online (new functionality in Version 11.70).

  Informix 11.70 provides the ability to upgrade the cluster online. The feature, called *rolling upgrade*, eliminates the downtime during the upgrade process.

13.2.1 Preparing a cluster for an upgrade

If you use a high availability cluster, you must coordinate the migration of all the servers that are involved in the cluster, and you must perform additional steps when preparing to migrate.

In the following section, we refer to the existing product installation as the *source server* and the new product installation as the *target server*.

To prepare the cluster for an upgrade, complete the following steps:

1. Prepare to upgrade the server.

   Preparing for migration includes gathering information about the current installation and backing up your data so that you can reinstall the previous version of the server and restore your data if you have a migration problem.

   For information about upgrading the server, refer to the Migrating and Upgrading section at the Informix Information Center at the following address:

   http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.mig.doc/ids_mig_040.htm

2. Install the target server.

   Install the target server in a different location than the source server. The target server must be installed on all the servers in the cluster.

3. Copy the configuration files (the `onconfig` and `sqlhosts` files) from the source server to the target server installation directory on all of the servers in the cluster.

4. Install any user-defined objects or DataBlade modules (that are used on the source server) onto all of the target servers in the cluster.

5. Back up your primary server. You can perform this step in either of the following ways:

   – Back up all logs, and then use **`onbar`** or **`ontape`** to make a level 0 backup of the primary server.

– Alternatively, if you have a high-availability cluster with a High Availability Data Replication (HDR) secondary server, you can use the HDR secondary server as a standby server for any contingencies that occur while you upgrade the primary server. However, if it is necessary to use an HDR secondary server as a standby server for contingencies, do not perform updates on the standby server while migration or reversion is in progress, because the updates cannot be replicated and will be lost. Additionally, nonlogged objects on the primary server will not exist on the secondary server.

  Do not use RSS servers as backup servers, because transactions could be lost.

13.2.2 Upgrading a cluster to a new fix pack or PID

If you are upgrading clusters from one fix pack or PID to a new fix pack or PID and the migration does not involve actual conversion or reversion, you must complete additional tasks when you upgrade. However, you do not need to recreate the secondary servers after you upgrade the primary database server.

The prerequisites for this type of upgrade are as follows:

1. Verify that you are upgrading to a new fix pack or PID in which standard conversion procedures are not necessary. If you are upgrading to a new fix pack or PID that requires you to complete standard conversion procedures, follow the steps listed in 13.2.3, “Upgrading a cluster to a new release” on page 510.
2. Complete the steps in 13.2.1, “Preparing a cluster for an upgrade” on page 508.
3. Perform all migration operations as the informix user.

Complete the following steps to upgrade clusters to a new fix pack or PID:

1. Stop the Connection Manager by running **`oncmsm -k connection_manager_name`**.
2. If you are using a High Availability Data Replication (HDR) secondary server as a backup server in case of contingencies:
   a. Quiesce the primary server by running **`onmode -sy`** to prevent user connections to the server.
   b. Force a checkpoint by running **`onmode –c`** on the primary server.
3. Stop secondary servers in the cluster in the following order:
   a. If you have Remote Stand-alone Servers (RSS), stop them by running **`onmode -ky`**.

   b. If you have Shared Disk Servers (SDS), stop them by running **`onmode -ky`**.
   c. If you have an HDR secondary server, stop it by running **`onmode -ky`**.
4. Stop the primary server by running **`onmode -ky`**.
5. On each server, set the INFORMIXDIR environment variable to the full path name of the target installation.
6. Ensure that all of the necessary configuration files are available in the target installation.

   Start the servers in the cluster and perform additional tasks in the following order:

   a. Start the primary server by running **`oninit`**.
   b. Wait for primary server to be in online (multi-user) mode.
   c. Start the Connection Manager by running **`oncmsm`**.
   d. Start the HDR secondary server by running **`oninit`**.
   e. Start SDS by running **`oninit`**.
   f. Start RSS by running **`oninit`**.

13.2.3 Upgrading a cluster to a new release

If you have a high availability cluster with one or more secondary database servers and are migrating to a new release, or if you are upgrading to a new fix pack or PID that requires you to complete standard conversion procedures, you must complete additional tasks when you migrate.

When you migrate clusters, you have to migrate only the primary database server.

Beginning with Informix 11.50xC6, the server automatically removes secondary servers when you migrate or revert. After migration or reversion on the primary server is complete, you must recreate all HDR, RSS, and SDS secondary servers in a high availability cluster.

The prerequisites for this type of upgrade are as follows:

1. Prepare to upgrade the server.

   Preparing for migration includes gathering information about the current installation and backing up your data, so that you can reinstall the previous version of the server and restore your data if you have an upgrade problem.

   For information about upgrading the server, refer to the Migrating and Upgrading section at the Informix Information Center at the following address:

   http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.mig.doc/ids_mig_040.htm

2. Complete the steps in 13.2.1, “Preparing a cluster for an upgrade” on page 508.
3. Perform all migration operations as the informix user.

When you migrate clusters, be sure to stop and start the servers in the cluster in the order shown in the following procedure.

To migrate to a new release with high availability clusters, complete the following steps:

1. Stop the Connection Manager by running **`oncmsm -k connection_manager_name`**.
2. If you are using an HDR secondary server as a backup server in case of contingencies:
   a. Quiesce the primary server by running **`onmode -sy`** to prevent user connections to the server.
   b. Force a checkpoint by running **`onmode –c`** on the primary server.
3. Stop the secondary servers in the cluster in the following order:
   a. If you have RSS servers, stop them by running **`onmode -ky`**.
   b. If you have SDS servers, stop them by running **`onmode -ky`**.
   c. If you have an HDR secondary server, stop it by running **`onmode -ky`**.
4. If you are migrating to pre-11.50.xC6 versions of Informix, perform the following tasks on the primary server:
   a. If you have an HDR pair, split the HDR pair by running **`onmode -d standard`**.
   a. If you have an RSS server, delete the RSS entries by running **`onmode -d delete RSS`** ***`rss_server_name`***.
5. Stop the primary server by running **`onmode -ky`**.
6. On each server, set the INFORMIXDIR environment variable to the full path name of the target server installation.
7. Ensure that all of the necessary configuration files are available in the target installation.
8. Start the primary server by running **`oninit`**.
9. Ensure that the conversion to the target server was successful and that the server is in multi-user mode.
10. Start the Connection Manager by running **`oncmsm`**.

11. If you are migrating from pre-11.10 versions of Informix and need SDS servers on the primary server in a shared-disk cluster, set the primary server by running **`onmode -d set SDS primary`** ***`primary_server_name`***.

12. Start the SDS servers by running **`oninit`**.

13. Start the servers in the cluster and perform additional tasks in the following order:

    b. Back up all logs. Then run **`onbar`** or **`ontape`** to make a level 0 backup on the primary server to reestablish the RSS and HDR servers if necessary.

    c. If you have RSS servers:

       i. Add RSS entries on the primary server by running **`onmode –d add RSS`** ***`rss_server_name`***.

       ii. Start RSS servers with level 0 restore operations from the level 0 backup that was made on the primary server after migration.

       iii. On the RSS servers, run **`onmode -d RSS`** ***`primary_server_name`*** and wait for the `RSS secondary server operational` message to appear after each command.

    d. If you have an HDR secondary server:

       i. Reestablish the pair on the primary server by running **`onmode -d primary`** ***`hdr_secondary_server_name`***.

       ii. Start the HDR secondary server with level 0 restore operations from the level 0 backup that was made on the primary server after migration.

       iii. On the HDR secondary server, run **`onmode -d secondary`** ***`primary_server_name`*** and wait for the H`DR secondary server operational` message to appear after each command.

13.2.4 Upgrading a cluster while it is online

Version 11.70 of the Informix database software provides the ability to upgrade a high availability cluster while it is online. This feature, called a *rolling upgrade*, eliminates down time during the upgrade process.

Informix database software versions earlier than 11.70 required that the cluster be shut down before upgrading the server software from earlier versions. The rolling upgrade feature allows you to upgrade the cluster with zero or minimal down time. In the rolling upgrade process, the primary server and one of the secondary servers (either the RSS or the HDR server) are transformed to form a pair of ER servers. ER has always supported rolling upgrades because it allows replication between heterogeneous server versions. By providing the ability to easily transform the HDR or RSS system into an ER system, it becomes possible to perform the rolling upgrade to an Informix 11.70 server.

Figure 13-8 illustrates the Informix high availability cluster where HDR is converted to ER.

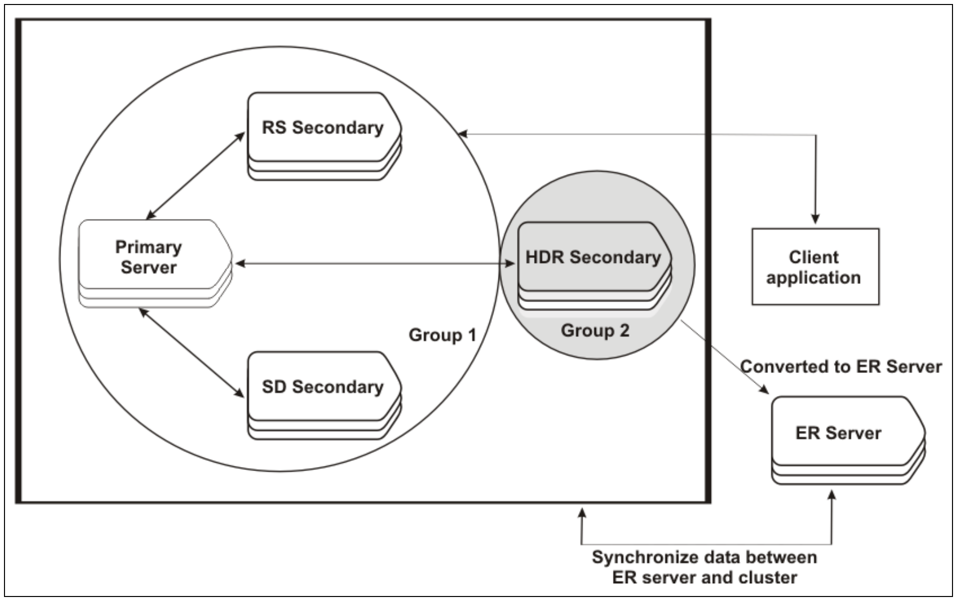


Figure 13-8 Informix high availability cluster where HDR is converted to ER

The rolling upgrade process starts with a check to see if the secondary server can be transformed into an ER server. This is achieved by using the cdr utility with the sec2er option, which is implemented in Informix 11.70. Replicates are automatically defined so that the data is synchronized between the primary server in the cluster and the transformed ER server.

Figure 13-9 illustrates the Informix high availability cluster synchronizing with ER.

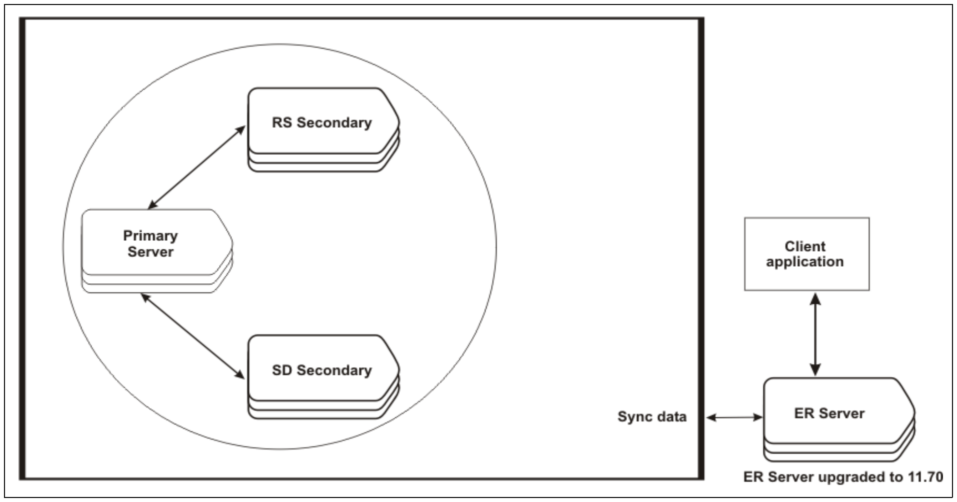


Figure 13-9 Informix high availability cluster synchronize with ER

After synchronization, the ER server can be upgraded to Informix 11.70. It can service the user connections while the servers of the high availability cluster are brought down. The upgraded ER server can be used to re-configure the cluster using Informix 11.70by using a new utility called ifxclone implemented in Informix 11.70. For more information about ifxclone, see Chapter 14, “Using ifxclone” on page 525.

Figure 13-10 illustrates the Informix high availability cluster using ifxclone.

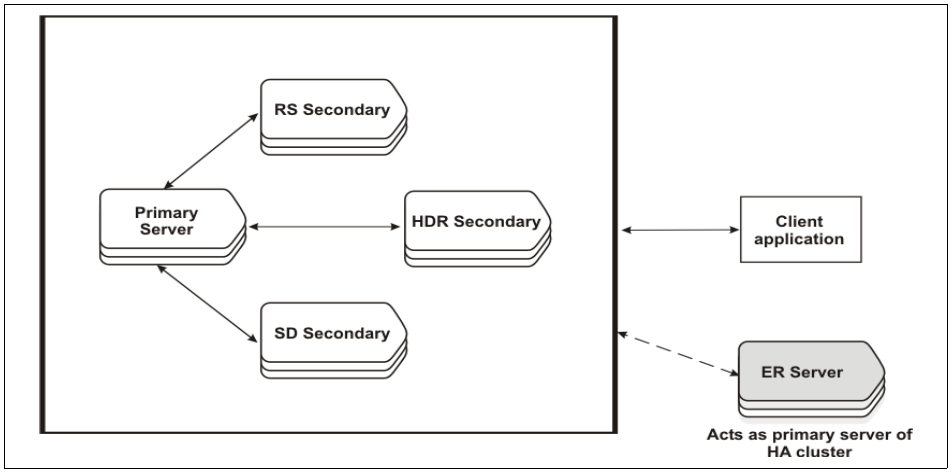


Figure 13-10 Informix high availability cluster using ifxclone

13.2.5 Approaches to consider

Here we show two approaches to perform a rolling upgrade. Based on the way a high availability cluster is set up and whether or not the Connection Manager is used to manage the user connection, you can choose one of the following options:

- Option 1: Upgrade clusters without Connection Manager

  User applications must take steps to move their connections to the appropriate servers during the upgrade process to avoid client connection downtime.

- Option 2: Upgrade clusters with Connection Manager and zero downtime during the upgrade

  If achieving zero downtime is paramount, then you can use this option as long as the Connection Manager infrastructure supports Connection Manager groups.

  In this option, you add a new Connection Manager instance to manage user connections while the cluster is upgraded. This option involves configuring a new 3.70.xC1 Connection Manager instance that supports ER and makes changes to the `sqlhosts` file (or other connection mechanisms) for user applications. If Connection Manager group support infrastructure is enabled, you only need to add the new Connection Manager for ER to your existing CM group. Adding Connection Manager to the group ensures that no user connections experience down time during the cluster upgrade.

Prerequisites

No matter which methods you choose to use, these prerequisites must be satisfied:

- Non-logged databases are not supported.
- Raw or unlogged tables are not supported.
- Typed tables are not supported unless the typed table contains a primary key.
- User defined types that do not support ER are not supported.
- For versions of Informix software earlier than 11.50xC7, converting a primary and secondary server pair to ER is not supported if a table does not have a primary key.
- For versions of Informix software earlier than 11.10, the **`sec2er`** command is not supported

Option 1: Upgrading clusters without Connection Manager

In the following example, the high availability cluster consists of a primary server, one SDS, one RSS, and an HDR secondary server. The HDR server is being transformed from the cluster and upgraded to Informix 11.70 as a part of the rolling upgrade process.

The upgrade steps are as follows:

1. Install Informix 11.70, if necessary.

   The cdr utility with the sec2er option, which is required to transform the HDR or RSS secondary server to ER server, is available with Informix 11.70 but not with Informix 11.50. So you must install Informix 11.70 on any machine that has remote trusted access to the cluster machines.

   The cdr utility with the sec2er option can be run against an Informix 11.50 server from an Informix 11.70 installation.

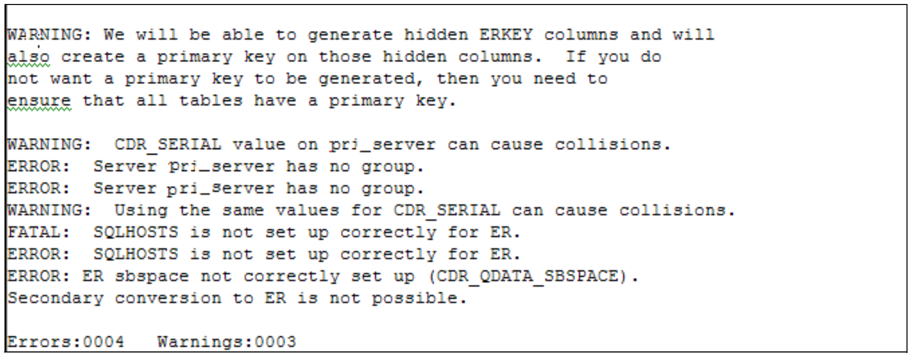
2. Check whether the HDR server can be converted to an ER server.

   As the informix user, run the following command against the 11.50 server:

   ```
   <Informix_1170_installdir>/bin/cdr check sec2er -c pri_server -p sec_server
   ```

   This command examines the primary and secondary nodes and determines if it is possible to perform a split. It displays warning of possible problems that might be encountered as well as error conditions that prevent a split. Optionally, it displays the output of the commands that can be executed to perform the split.

Figure 13-11 shows the possible errors and warning that you might see while executing **cdr check sec2er**.



```
WARNING: We will be able to generate hidden ERKEY columns and will
also create a primary key on those hidden columns.  If you do
not want a primary key to be generated, then you need to
ensure that all tables have a primary key.

WARNING:  CDR_SERIAL value on pri_server can cause collisions.
ERROR:  Server pri_server has no group.
ERROR:  Server pri_server has no 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
```

Figure 13-11   Possible errors and warning

The errors shown above can be fixed as follows:

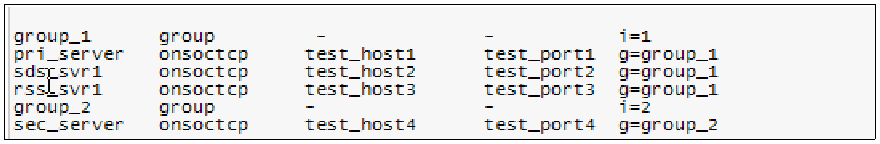
– Missing ER spaces in CDR_QDATA_SBSPACE:

  Add smartblob spaces (for example, er_sbspace) and set the **onconfig** parameter CDR_QDATA_SBSPACE to er_sbspace on the primary server and the server being transformed (HDR in this example). ER uses these smartblob spaces to store the spooled transaction row data.

– Missing groups in SQLHOSTS

  ER requires that all database servers participating in the replication must be a member of database server group. Define the server groups in the `sqlhosts` file. The secondary server being transformed should belong to a different ER group.

  Update the `sqlhosts` file. Figure 13-12 shows an example of an updated `sqlhosts` file.

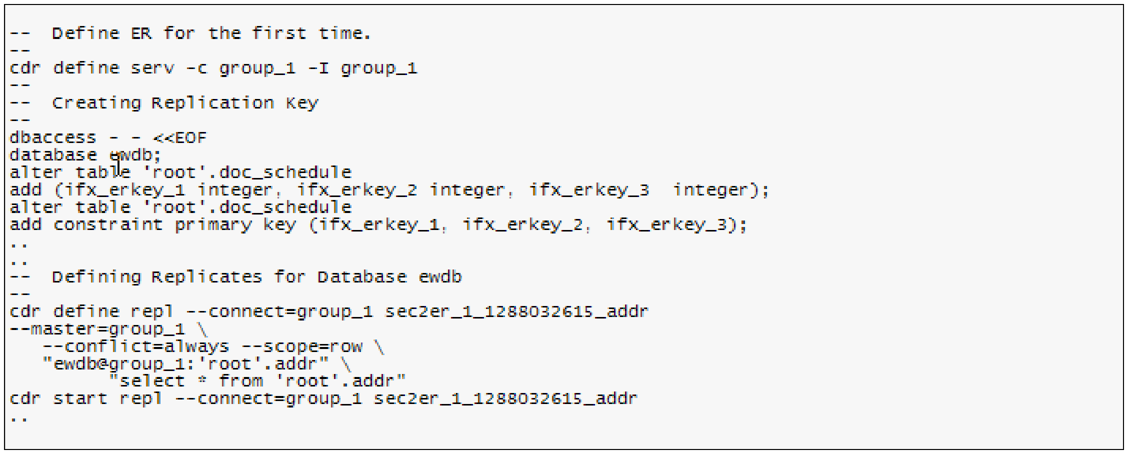


```
group_1      group         -            -            i=1
pri_server   onsoctcp     test_host1   test_port1   g=group_1
sds_svr1     onsoctcp     test_host2   test_port2   g=group_1
rss_svr1     onsoctcp     test_host3   test_port3   g=group_1
group_2      group        -            -            i=2
sec_server   onsoctcp     test_host4   test_port4   g=group_2
```

Figure 13-12   High availability clusters after adding logical log groups

Re-run **cdr check sec2er** again after fixing the errors to ensure that all the errors are resolved.

Figure 13-13 shows the output from **`cdr check sec2er`** after the errors are fixed.



```
--  Define ER for the first time.
--
cdr define serv -c group_1 -I group_1
--
--  Creating Replication Key
--
dbaccess - - <<EOF
database ewdb;
alter table 'root'.doc_schedule
add (ifx_erkey_1 integer, ifx_erkey_2 integer, ifx_erkey_3  integer);
alter table 'root'.doc_schedule
add constraint primary key (ifx_erkey_1, ifx_erkey_2, ifx_erkey_3);
..
..
--  Defining Replicates for Database ewdb
--
cdr define repl --connect=group_1 sec2er_1_1288032615_addr
--master=group_1 \
   --conflict=always --scope=row \
   "ewdb@group_1:'root'.addr" \
         "select * from 'root'.addr"
cdr start repl --connect=group_1 sec2er_1_1288032615_addr
..
```

Figure 13-13   Output from the cdr check sec2er command after the errors are fixed

Note: Enterprise Replication requires that primary keys be defined on all tables. If any of the tables are missing primary keys, those tables are altered to add ER shadow column keys that act as primary keys. As you can see from the output in Figure 13-13, the **`cdr start sec2er`** command defines an ER server, creates ER shadow columns on tables that do not have them, and then creates and starts replicates for all tables in the user database.

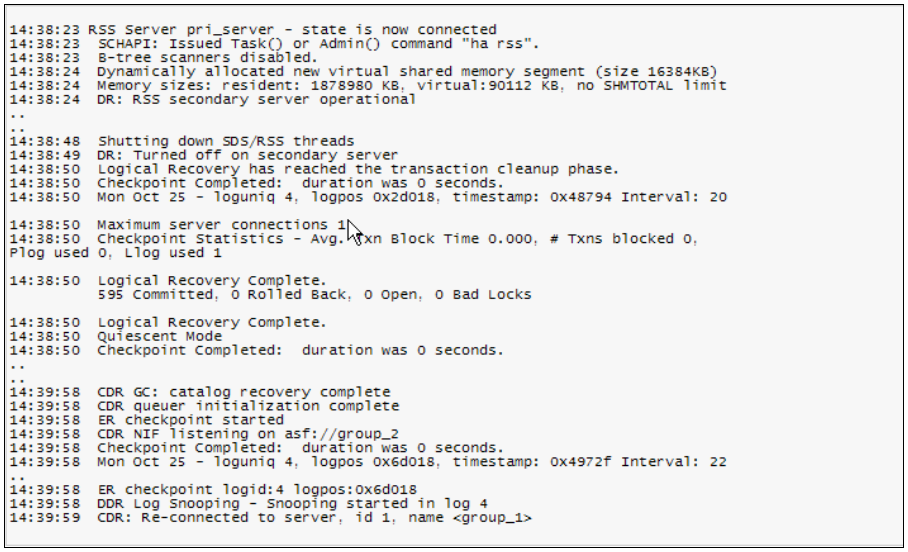
3. Transform the secondary server to an ER server.

   In this example, we are converting the HDR secondary server to an ER server. As the Informix user, run the following command:

   ```
   /<Informix_1170_installdir>/bin/cdr start sec2er -c pri_server sec_server
   ```

   After you successfully run the command, the HDR is converted to an ER server. If any of the tables are missing a primary key, the **`cdr start sec2er`** command creates an ER key shadow column that acts the primary key. The **`cdr start sec2er`** command ensures that replicates are automatically created on all tables in the user databases so that data is synchronized with the cluster.

Figure 13-14 shows an example of the message log from the HDR secondary server.



```
14:38:23 RSS Server pri_server - state is now connected
14:38:23  SCHAPI: Issued Task() or Admin() command "ha rss".
14:38:23  B-tree scanners disabled.
14:38:24  Dynamically allocated new virtual shared memory segment (size 16384KB)
14:38:24  Memory sizes: resident: 1878980 KB, virtual:90112 KB, no SHMTOTAL limit
14:38:24  DR: RSS secondary server operational
..
..
14:38:48  Shutting down SDS/RSS threads
14:38:49  DR: Turned off on secondary server
14:38:50  Logical Recovery has reached the transaction cleanup phase.
14:38:50  Checkpoint Completed:  duration was 0 seconds.
14:38:50  Mon Oct 25 - loguniq 4, logpos 0x2d018, timestamp: 0x48794 Interval: 20

14:38:50  Maximum server connections 1
14:38:50  Checkpoint Statistics - Avg. Txn Block Time 0.000, # Txns blocked 0,
Plog used 0, Llog used 1

14:38:50  Logical Recovery Complete.
          595 Committed, 0 Rolled Back, 0 Open, 0 Bad Locks

14:38:50  Logical Recovery Complete.
14:38:50  Quiescent Mode
14:38:50  Checkpoint Completed:  duration was 0 seconds.
..
..
14:39:58  CDR GC: catalog recovery complete
14:39:58  CDR queuer initialization complete
14:39:58  ER checkpoint started
14:39:58  CDR NIF listening on asf://group_2
14:39:58  Checkpoint Completed:  duration was 0 seconds.
14:39:58  Mon Oct 25 - loguniq 4, logpos 0x6d018, timestamp: 0x4972f Interval: 22
..
14:39:58  ER checkpoint logid:4 logpos:0x6d018
14:39:58  DDR Log Snooping - Snooping started in log 4
14:39:59  CDR: Re-connected to server, id 1, name <group_1>
```

Figure 13-14 Message log from HDR secondary server

An ER server is now established. ER supports replication between heterogeneous server versions. The original cluster with the primary, SDS, and RSS servers is running Informix 11.50, and the new ER server (sec_server) is also running Version 11.50.

4. Upgrade the ER server.

   Bring down the ER server and upgrade it to Informix 11.70. The client applications can continue to run against the 11.50 cluster while the ER server is upgraded.

   The steps for upgrading the ER server are as follows:

   a. Ensure that all logs are backed up by running the following command:

      ```
      ontape -a
      ```

   b. Stop the replication between ER server and cluster servers by running the following command:

      ```
      cdr stop
      ```

   c. Bring down the ER server by running the following command:

      ```
      onmode -kuy
      ```

   d. Bring up the Informix 11.70 server. Wait for the `Conversion Completed Successfully` message on the Informix 11.70 server.

e. Convert the syscdr database used by ER by running the following command:

```
sh concdr.sh 11.50 11.70
```

f. Start ER again on the upgraded ER server by running the following command:

```
cdr start
```

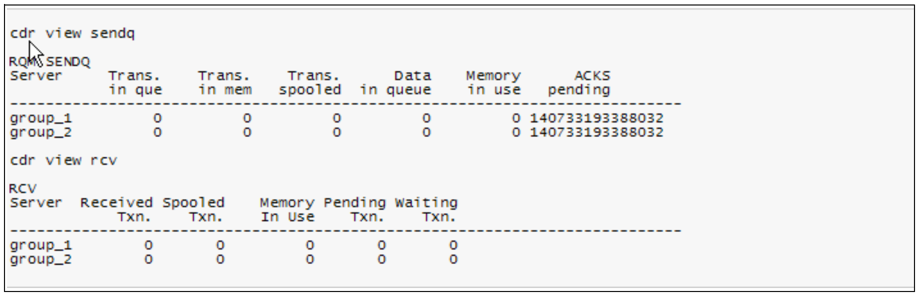
The ER server is now running Informix 11.70 and replicating with the cluster, which is still running Informix 11.50.

5. Synchronize the ER server with the cluster.

   Monitor the following commands to ensure that all data is synchronized between the upgraded ER server and the cluster:

   ```
   cdr view sendq
   cdr view rcv
   ```

   After synchronization, the cdr send and receive queues should look like the ones shown in Figure 13-15.



```
cdr view sendq

RQM SENDQ
Server     Trans.    Trans.    Trans.      Data    Memory      ACKS
           in que    in mem   spooled  in queue    in use   pending
---------------------------------------------------------------------------
group_1         0         0         0         0         0 140733193388032
group_2         0         0         0         0         0 140733193388032

cdr view rcv

RCV
Server  Received Spooled    Memory Pending Waiting
            Txn.    Txn.    In Use    Txn.    Txn.
---------------------------------------------------------------------------
group_1        0       0         0       0       0
group_2        0       0         0       0       0
```

Figure 13-15 Sample output after data synchronization

6. Stop the cluster

   Stop the cluster and move the application to the new ER server (sec_server).

7. Remove ER.

   You can now remove ER, which was temporarily used to facilitate the upgrade of your high availability cluster. Run the following command:

   ```
   cdr delete server group_2
   ```

8. Set up the cluster again.

   Set up the cluster from the ER server again, which now acts as the new primary server of the cluster.

Informix 11.70 provides a new utility called ifxclone that can be used to clone a server with minimal setup. Here are the steps for setting up the cluster using ifxclone utility:

- Set up RSS with ifxclone:
  1. Set ENABLE_SNAPSHOT_C0PY in the source server. Run the following command on the source server (the new ER server in this example):

     ```
     onmode -wf ENABLE_SNAPSHOT_COPY=1
     ```

  2. Ensure LOG_INDEX_BUILDS is set to 1 in the source server configuration.
  3. Define the intended RSS server on the source server by running the following command:

     ```
     onmode -d add RSS rss_svr1
     ```

  4. As the informix user, run the following command on the target to set up RSS:

     ```
     ifxclone -T -L -S sec_server -I test_host4 -P test_port4 -t rss_svr1
     -i test_host3 -p test_port3 -d RSS
     ```

- Set up HDR using ifxclone:
  1. Run the ifxclone utility on the intended target for HDR secondary server as the informix user:

     ```
     ifxclone -T -L -S sec_server -I test_host4 -P test_port4 -t
     pri_server -i test_host1 -p test_port1 -d RSS
     ```

     Now pri_server is an RSS server.
  2. Promote the RSS server to an HDR secondary server by running the following command:

     ```
     onmode -d secondary sec_server
     ```

Option 2: Upgrading clusters with Connection Manager with zero downtime during the upgrade

In the following example, the high availability cluster consists of a primary server, one SDS server, one RSS server, and an HDR secondary server. The Connection Manager (CM) is also set up with Connection Manager groups. The RSS server is being transformed and upgraded to an Informix 11.70 server as a part of the rolling upgrade process.

Step 5 and 6 add a new Connection Manager for ER known as *roll_upgrade_er*, which re-routes the Informix clients from transaction processing to the new ER server created in Step 3.

The sample sqlhosts and the connection manager configuration files Figure 13-16 and Figure 13-17 as a reference.

```
#Primary Server
pri_server onsoctcp        test_host1      test_port1
#SDS server
sds_svr1   onsoctcp        test_host2      test_port2
# RSS Server
rss_svr1   onsoctcp        test_host3      test_port3
# HDR Secondary server
sec_server onsoctcp        test_host4     test_port4
#These are Connection managers group
cm_group1         group                 -              -        i=100
cm_load         onsoctcp        test_host3    cmport_load    g=cm_group1
cm_report       onsoctcp        test_host2    cmport_report   g=cm_group1
```

Figure 13-16 Sample sqlhosts file

```
DEBUG           1
# Connection Manager Name
NAME            roll_upgrade_mach1
# re-route Informix for transaction processing
SLA             cm_load=(HDR+RSS+SDS+PRI)
# Failover Configuration
FOC     SDS,0
# worker threads for each SLA listener, default is 8
SLA_WORKERS     10
# Connection Manager Message file
LOGFILE         /work/logs/roll_cm1.log
```

Figure 13-17 Sample CM configuration file cm_set.1cfg

The upgrade steps are as follows:

1. Install Informix 11.70, if necessary.

   Refer to step 1 on page 516.

2. Check whether the RSS serve can be converted to an ER server.

   As the informix user, run the following command:

   ```
   /<Informix_1170_installdir>/bin/cdr check sec2er -c pri_server -p rss_svr1
   ```

3. Transform the secondary server to an ER server.

   As the informix user, run the following command:

   ```
   /<Informix_1170_installdir>/bin/cdr start sec2er -c pri_server rss_svr1
   ```

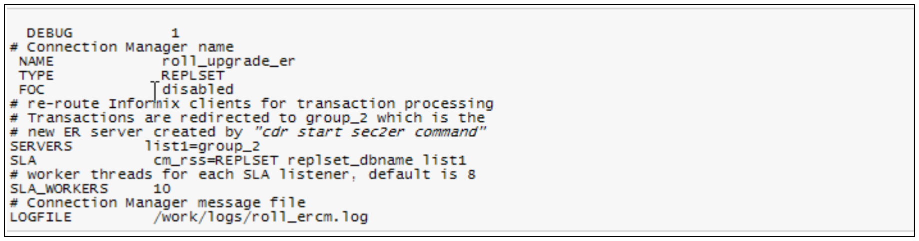
   After successfully running the command, the RSS is converted to an ER server. If any of the tables are missing a primary key, the **`cdr start sec2er`** command creates an ER key shadow column that acts the primary key. The **`cdr start sec2er`** command ensures that replicates are automatically created on all tables in the user databases so that it can sync up the data with the cluster.

4. Upgrade the ER Server.

   Refer to step 4 on page 519.

5. Define a new Connection Manager for ER.

Figure 13-18 shows a sample Connection Manager configuration file.

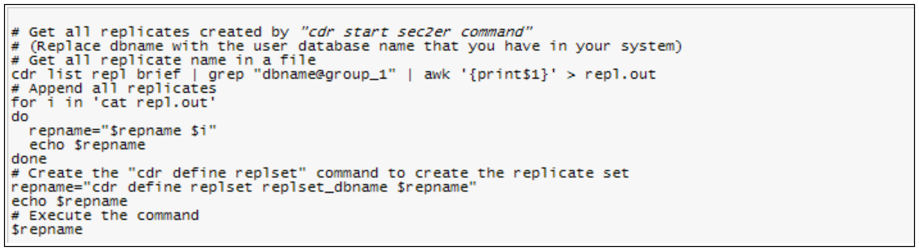


```
 DEBUG          1
# Connection Manager name
 NAME           roll_upgrade_er
 TYPE           REPLSET
 FOC            disabled
# re-route Informix clients for transaction processing
# Transactions are redirected to group_2 which is the
# new ER server created by "cdr start sec2er command"
SERVERS        list1=group_2
SLA             cm_rss=REPLSET replset_dbname list1
# worker threads for each SLA listener, default is 8
SLA_WORKERS     10
# Connection Manager message file
LOGFILE         /work/logs/roll_ercm.log
```

Figure 13-18 Sample Connection Manager configuration file

The Connection Manager for ER requires a replicate set to be defined (in the sample configuration file, the replicate set used is *replset_dbname*). The **`cdr start sec2er`** command automatically creates replicates for all tables in the user database. The Connection Manager expects a replicate set. A replicate set combines many replicates to form a set that can be administered as a single unit.

Figure 13-19 shows an UNIX shell script to create replicate set from the list of replicates.



```
# Get all replicates created by "cdr start sec2er command"
# (Replace dbname with the user database name that you have in your system)
# Get all replicate name in a file
cdr list repl brief | grep "dbname@group_1" | awk '{print$1}' > repl.out
# Append all replicates
for i in 'cat repl.out'
do
  repname="$repname $i"
  echo $repname
done
# Create the "cdr define replset" command to create the replicate set
repname="cdr define replset replset_dbname $repname"
echo $repname
# Execute the command
$repname
```

Figure 13-19 UNIX shell script to create a replicate set from the list of replicates

Execute the following command to list all the replicates defined in the replicate set - replset_dbname:

```
cdr list replicateset replset_dbname
```

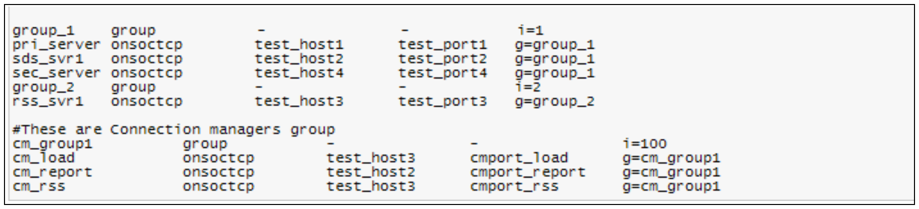
6. Start the new Connection Manager for ER Group.

   Execute the following command:

   ```
   oncmsm -c /work/scripts/cmsm_er.cfg
   ```

   Add the new SLA in the `sqlhosts` file under the connection group.

Figure 13-20 shows the updated `sqlhosts` file with the new SLA.



```
group_1    group         -            -           i=1
pri_server onsoctcp      test_host1   test_port1  g=group_1
sds_svr1   onsoctcp      test_host2   test_port2  g=group_1
sec_server onsoctcp      test_host4   test_port4  g=group_1
group_2    group         -            -           i=2
rss_svr1   onsoctcp      test_host3   test_port3  g=group_2

#These are Connection managers group
cm_group1          group         -            -              i=100
cm_load            onsoctcp      test_host3   cmport_load    g=cm_group1
cm_report          onsoctcp      test_host2   cmport_report  g=cm_group1
cm_rss             onsoctcp      test_host3   cmport_rss     g=cm_group1
```

Figure 13-20 Updated sqlhosts file with the new SLA

7. Stop the cluster.

   Stop the cluster on the Informix 11.50 server. Any client connection connecting to the Connection Manager group (cm_group1 in the example) will be redirected to the ER server (rss_svr1 through the SLA cm_rss, which is defined in the CM group in `sqlhosts`).

8. Set up the cluster again.

   Refer to step 8 on page 520.

9. Stop the Connection Manager for ER group.

   After MACH11 cluster is up and running, stop the Connection Manager for ER group.

   You can now remove ER, which was temporarily used to facilitate the upgrade of your high availability cluster. Run the following command:

   ```
   cdr delete server group_2
   ```

   You can also remove the newly added entry from the Connection Manager group in `sqlhosts` that corresponds to the ER server (cm_rss).

   Note that your original HDR and RSS servers are transformed to an ER server and upgraded to Informix 11.70. The upgraded ER server acts as the primary server and can be used as source server to rebuild your other cluster servers by cloning. If you choose to go back to your original cluster configuration, you can plan the appropriate failover operations.

14 Using ifxclone

The flexible network of the modern infrastructures ties various types of Informix servers together in a virtualized environment, a MACH11 cluster, or in a cloud. Having the independent stand-alone database servers distributed across remote company locations or stores is a common scenario in an enterprise today. The number of database servers required varies depending on the application workload, used locations, number of subsidiaries, and the degree of securing data availability.

In addition to the existing MACH11 SDS solution that enables you to run a flexible number of Informix database servers sharing the same storage, Informix 11.70 introduces the *ifxclone* utility. Using ifxclone, DBAs are able to create on demand copies of a master database server and distribute the copies to the appropriate target host machines automatically. The new database server copies can be set up as different server types.

In this chapter, we provide a detailed overview of the functionality and setup requirements of the ifxclone utility. We focus our discussion on the common use case for expanding the existing database environments with different target server types.

14.1 General considerations for using ifxclone

The Informix database cluster and the loosely network-tied database server environments consist of more and more participating servers. Effective resource management becomes a key requirement in a company. The following factors are the influential factors for the increasing number of server resources in an database server production environment:

- Increased requirements in data security and high availability:
  - Spending more resources for failover to ensure critical data
  - Adding standby servers as a reaction for the increased occurrences of natural disasters, such as floods or earthquakes.
- Spending more money in leasing hardware on a time-share basis for processing high peak loads.
- Applications require dedicated systems for specific data subsets, such as local data, time based data, static data, or test environments for application development.
- Spending more effort in using “virtual” resources.

Informix provides several core functionalities for DBAs to set up a new database server manually, including backup and restore, external tables, and hardware mirror splitting. However, these approaches all use a combination of the identification, extraction, and application of deltas that happen during the physical load, so they cannot be automated.

Ifxclone provides a flexible and simple solution to fast clone a system with a single point of execution.

14.1.1 Ifxclone basics

Using the ifxclone utility requires that you identify a master database server instance that can be copied to a different target machine. The target machine can be a physical system or a VMware image that is identified by an unique IP address. The copy process is based on a backup of the master database server and requires the same database version and same operating system on both the master and target systems.

The ifxclone utility supports copying the master database server into the following types of target servers:

- Standard stand-alone server
- RSS servers
- ER and Flexible Grid participating server

In the following sections, we describe the clone process in detail based on the use cases.

14.1.2 Prerequisites for executing ifxclone

In this section, we discuss the tasks required before we start the cloning of the primary database server, the parameter list of the utility, and the changes required on both the primary and the target database servers.

Parameter values for ifxclone

Example 14-1 shows the relevant parameters of ifxclone.

Example 14-1 ifxclone parameter list

```
$ ifxclone
-S      --source=<name>                 # Name of the source node
-I      --sourceIP=<IP>                 # IP address of source node
-P      --sourcePort=<port>             # Port number of source server
-t      --target=<name>                 # Name of target server
-i      --targetIP=<IP>                 # IP address of target server
-p      --targetPort=<port>             # Port number of target server
-d      --disposition=[RSS|ER]         # Clone disposition (default:standard)
-s      --size=[tiny|small|medium|large]  # Configuration size
-c      --configParm="PARAMETER=VALUE"    # Configuration override
-L      --useLocal                      # Use the local config and sqlhost
-T      --trusted                       # No userid/password required
```

The information that must be collected ahead of time for the ifxclone parameters is as follows:

- From the master and the target servers:
  - The database server name of the original and the target database servers.
  - The IP addresses of the host machine where both database servers reside.
  - The port numbers where the database server are or will be configured on.
- Informix configuration file:

  The `onconfig` file and the `sqlhosts` file are required.
  - On UNIX, you can either specify your own local copy of the `onconfig` file and the `sqlhosts` file for the target or let ifxclone create a new instantiation of the master files. Creating your own files gives you the flexibility to adjust the appropriate parameters according the needs of the new instance.

- On Windows, to use the local copy, you must create a new empty instance on your target machine using the server instance manager. Then you must add a connection entry for the target server using the setnet32 utility.
- When embedding the clone process, ifxclone creates the local configuration files based on the master configuration. The local adjustments can be applied by the specification of one or more -c parameters.

Example 14-2 shows an **ifxclone** command example. In this example, the master and target Informix database server are on different machines. We use a trusted environment without asking for a password, which is common in an embedded scenario. Additionally, we let **ifxclone** copy the source configuration files to the target and define specific replacements for some of the **onconfig** parameters.

Example 14-2 Sample call of ifxclone with various input parameters

```
$ifxclone -S primary_dbserver -I remotehost -P 20020 -t new_clone -i localhost
-p 20021 -T -c "SERVERNUM=101" -c "MSGPATH=/work/1170FC2B1/tmp/newclone.log"
```

Database server configuration

Before starting the clone process, there are administration tasks that must be done ahead of time on the primary and target servers.

Primary database server

On the primary database server where the snapshot is taken, you must set the ENABLE_SNAPSHOT_COPY **onconfig** parameter to allow ifxclone to initiate the copy process using one of the following methods:

- On the shell, run the following command to enable the snapshot copy dynamically for the server:

  ```
  onmode -wm ENABLE_SNAPSHOT_COPY=1
  ```

- Set the value in the `onconfig` file and restart the database server.

Target database server

In the target database server, complete the following steps:

1. Create each chunk file on the target database server with that same layout that is on the primary server host.
   - If cooked files are used, create empty files.
   - If you cannot have the same file system layout for your chunks on the target host machine, solve the layout difference with symbolic links.
2. Ensure that your `onconfig` file has the FULL_DISK_INIT parameter set to 1.

If you are cloning to a system with existing Informix database server, this parameter enables ifxclone to overwrite the existing copy. This change is important in an automated or embedded environment. The change must be applied again in case the copy process is repeated, because after server initialization, the value of FULL_DISK_INIT is reset to prevent the server from overwriting.

14.2 Creating a new standard server

In this section, we introduce cover two infrastructures for adding a new independent standard server:

- The newly cloned server contains data that is updated for the testing and development purposes.
- The newly closed server contains static data that refers to a particular reporting time stamp and should not be updated.

14.2.1 Cloning an application development server

Cloning a new standard server from a primary database server is a common request in an environment where you have ongoing application development projects that must use the actual data but are not allowed to access the production environment. You can set up a trigger to clone the development system from the primary server periodically.

The appropriate infrastructure is illustrated in Figure 14-1.

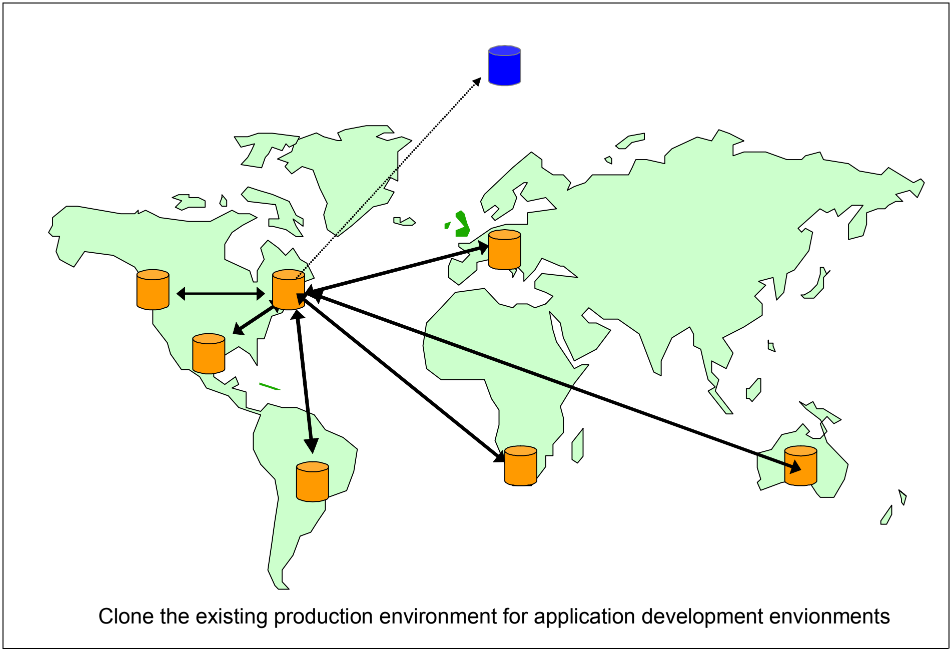

Clone the existing production environment for application development envionments

Figure 14-1 Adding a new standard server to a database server infrastructure

14.2.2 Spawning a data mart for reporting purposes

Another possible use case for cloning a system using ifxclone is to spawn your current production data at the end of certain time period, such as on a monthly or quarterly base, for the reporting system. The production environment continues to be updated. The spawned database server is frozen at the time stamp of the spawn and can be used for reporting purposes.

Figure 14-2 shows this scenario.

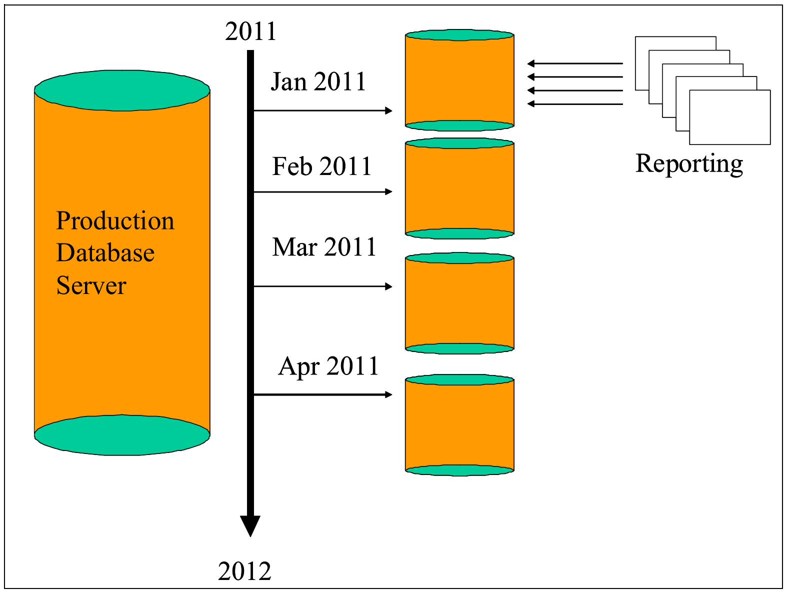


Figure 14-2 Spawning a data mart reporting server

14.2.3 Processing the clone operation

To perform a successful clone operation into a standard database server, complete the following steps:

- Identify the DBSERVERNAME for the primary and target database server.
- Identify the ports for both database servers.
- Identify the IP addresses for the host machines where the server reside.
- If you want to use a local copy of the `onconfig` and the `sqlhosts` file, set up the configuration files according to your needs.
- If you let ifxclone copy the configuration files, identify the necessary changes required and specify them in the parameter list.
- Make sure that you are able to access the machine where the primary database resides from the system that hosts the clone.
- Start the ifxclone utility as the user informix on the system where the clone should be set up.

Example 14-3 shows two **ifxclone** command examples with various parameters for creating a new standard Informix database server.

Example 14-3 Possible parameter lists for creating a new standard Informix server

```
#Trust the user, use a local copy of the configuration
$ifxclone -S primary -I primary_host -P 20020 -t clone -i target_host -p 20021
-T -L

#User and password need to be spec during exec , configuration files are taken
#from primary , MSGPATH is changed according to the local requirements
$ifxclone -S primary -I primary_host -P 20020 -t clone -i target_host -p 20021
-c "MSGPATH=/sqldists/1170FC2B1/tmp/clone1.log"
```

For copying the primary database server to a new standard server, the -d option is not required because it is the default.

14.3 Cloning an RSS server

In addition to cloning a standard Informix database server, the Ifxclone utility also supports cloning an RSS server. The major goals of cloning a system in an MACH11 cluster server include:

- Building a new MACH11 cluster by cloning a stand-alone primary server.
- Increasing the failover capabilities by adding new secondaries to the existing cluster.
- Offering the ability to add database server resources to serve peak time database application requirements.
- Offering the ability to add a database server for workload balancing.
- Separating OLTP and data warehouse applications to serve different resource requirements.

Figure 14-3 shows one example of using ifxclone for adding additional RSS servers to a cluster serving various types of applications. The goal is to separate the applications classified by their requirements on resources such memory and CPU utilization.

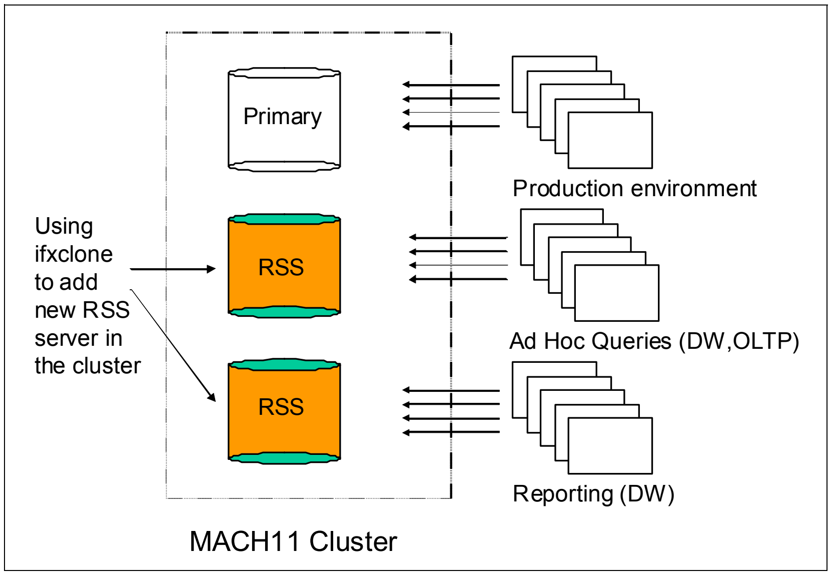


Figure 14-3 Adding an RSS server to a MACH11 cluster

14.3.1 Executing the clone operation

The prerequisites describe 14.1.2, “Prerequisites for executing ifxclone” on page 527 apply to cloning an RSS server.

To clone an RSS server, you must use the -d RSS option.

Example 14-4 shows an **ifxclone** command for cloning an RSS server. The **ifxclone** command must be run from the newly created target database server. The target database server host machine must have a connection to the host of the primary database server. The user running **ifxclone** is indicated as trusted and the local configuration files are used.

Example 14-4 ifxclone for an RSS secondary

```
ifxclone -S rss_new_or_existing_primary -I rssprim_host -P 20022 -t
rss_secondary -i localhost -p 20023 -T -L -d RSS
```

Adding a new RSS server using **ifxclone** does not require any activity on the primary server. The new secondary server will be added automatically.

You can monitor the execution status of the clone operation from the log file on both database servers. Example 14-5 shows the log file entries for a successful execution.

Example 14-5 Monitoring the success of an RSS secondary clone operation

```
onstat -g rss
RSS Server information:

RSS Srv      RSS Srv      Connection     Next LPG to send       Supports
name         status       status           (log id,page)        Proxy Writes
erclone1     Active       Connected               4,3088        N

#logfile on the new RSS secondary database server
20:40:00  Maximum server connections 0
20:40:01  DR: new type = RSS
20:40:01  RSS Server erclone - state is now connected
20:40:01  Logical Recovery Started.
20:40:01  Start Logical Recovery - Start Log 4, End Log ?
20:40:01  Starting Log Position - 4 0xc02278
20:40:01  Clearing the physical and logical logs has started
20:40:06  DR: RSS Secondary server operational

#logfile on the primary database server
20:39:44  Started Snapshot copy on rootdbs
20:39:57  Started Snapshot copy on s9_sbspc
20:40:00  Snapshot copy on s9_sbspc Completed.
20:40:00  Snapshot copy on rootdbs Completed.
20:40:01  RSS Server erclone1 - state is now connected
```

14.4 Expanding a grid or Enterprise Replication environment

The following examples demonstrate where adding new servers into a Flexible Grid is required:

- Opening a new location or subsidiary in a town or a country. A new system is required to take over the general inventory, product descriptions, and customer numbers.
- Opening a new office or a regional headquarters. A new system is required to manage people, projects, and assets.
- Taking over another company, and data consolidation is required.

We demonstrate adding new database servers to a grid using an example of a company that opens new remote locations. There is a master inventory server containing the base data set. All remote database servers need to maintain the master data set, but may manage locally owned data. New administration requirements, such as adding storage, memory, or schema rollouts, are triggered by the master database server. Figure 14-4 shows the new infrastructure.

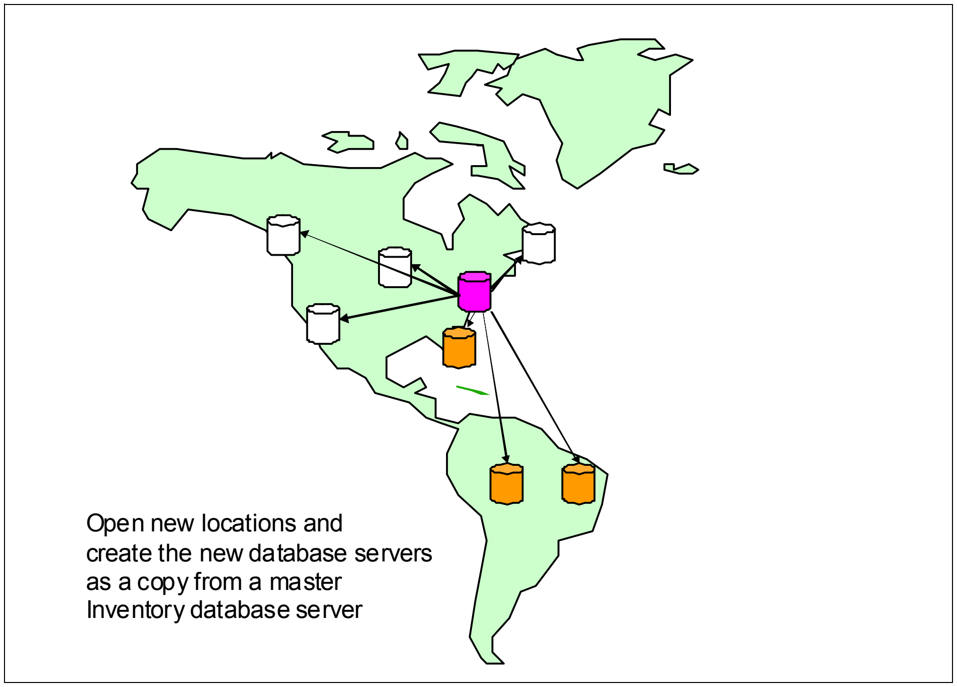


Figure 14-4 Roll out a master inventory to new locations

Prior to the introduction of ifxclone, expanding an existing Enterprise Replication (ER) base server environment was an manual process and had a certain administrative impact. First, you had to take and apply a database backup to the newly added database server. Second, you had to check if there is delta data from when the backup was taken to the present day. You had to set up a mechanism to identify and apply the delta on the target machine and then set up the ER infrastructure.

Using ifxclone, the DBA can easily set up an additional ER or Flexible Grid database server with point-in-time data that is ready to use without manual interaction.

14.4.1 Expanding the Flexible Grid

There prerequisites described in 14.1.2, “Prerequisites for executing ifxclone” on page 527 apply to expanding an existing Flexible Grid database server network using ifxclone.

In addition to the common prerequisites, to clone a new database server instance that participates in an ER or grid environment, the primary database server must be initialized as an ER database server and be part of an existing ER or grid infrastructure. Otherwise, the **ifxclone** command return the following error:

```
The server 'newgrid' cannot be added to the ER domain, ER error code 62.
For more details, see server log at server 'master'
or, if a leaf server, at its ER parent server.
```

For the ifxclone parameters, you have to specify the -d ER option. Example 14-6 shows how to clone an ER secondary.

Example 14-6 Clone an ER secondary

```
ifxclone -S master -I rootnode_host -P 20022 -t newgrid -i targetnode_host -p
20023 -T -L -d ER
```

In this example, we clone a database server and initialize the new server as an ER server. The server is automatically attached to the primary node. You must run **ifxclone** from the newly created target database server. The target database server host machine must have a connection to the host of the primary database server. The user running **ifxclone** is indicated as trusted and the local configuration files are used.

During the clone operation, the new server pair is changed to an RSS pair first, the data is applied to the target server, and any possible changes are tracked in the log file. After the new database server is successfully and physically established, the log files are replayed between the temporary RSS pair until the current time stamp. The new server is then changed to an ER server and is also ready to be used in a Flexible Grid environment.

You can view the clone operation status logged log file on both database servers. Example 14-7 shows a snapshot of the relevant messages during the execution of **ifxclone** on the primary and the new ER server.

Example 14-7 Message log entries when cloning an ER database server

```
#Primary database server
20:48:03  Started Snapshot copy on rootdbs
20:48:11  Started Snapshot copy on s9_sbspc
20:48:13  Snapshot copy on s9_sbspc Completed.
```

```
20:48:13  Snapshot copy on rootdbs Completed.
20:48:14  RSS Server newgrid - state is now connected
20:49:22  SMX thread is exiting because it received a CLOSE message
20:49:25  Invoking internal startup of ER
20:49:29  CDR NIF listening on asf://master
20:49:29  Internal stopping and restarting of ER was successful
20:49:29  ER clone:  Successfully created new ER node with id 2003 ('newgrid')
20:49:34  CDR: Re-connected to server, id 2003, name <newgrid>

#New ER database server
20:48:14  DR: new type = RSS
20:48:14  RSS Server erclone - state is now connected
20:48:19  DR: RSS secondary server operational
20:48:20  ER clone:  Starting split from primary server 'master'.
          For progress indication, also examine the message log file at
          the primary server.
20:48:20  ER clone: Joining this server into the ER domain.
20:48:21  ER clone: Determining Grid participation.
20:48:22  ER clone: Adding this server to ER replicates.
20:49:23  Shutting down SDS/RSS threads
20:49:30  ER clone:  Starting Enterprise Replication.
20:49:34  CDR NIF listening on asf://newgrid
20:49:34  ER checkpoint started
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publication provide additional information about the topic in this document:

- *Informix Dynamic Server 11: Extending Availability and Replication*, SG24-7488

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- *Availability Solutions with IBM Informix Dynamic Server 11.10,* found at:

  http://www.iiug.org/library/ids_11/whitepaper/ids11-availability-wp.pdf
- *IBM Informix Administrator's Guide*, SC27-3526
- *IBM Informix Dynamic Server Enterprise Replication templates, May 2005*, found at:

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0505kedia/
- *IBM Informix Enterprise Replication Guide*, SC27-3539

Online resources

This website is also relevant as a further information source:

- IBM Informix Information Center

  http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.start.doc/welcome.htm

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

IBM Informix Flexible Grid: Extending Data Availability

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Informix Flexible Grid

Extending Data Availability

Dynamic configuration and faster performance with Flexible Grid

Less and easier administration and higher availability

Load balancing and failover strategies

In this IBM Redbooks® publication, we focus on, and provide an overview of, the high availability and Enterprise Replication features of IBM Informix 11.70. Informix provides solutions for making data highly available in the MACH11 cluster. The components of the MACH11 cluster include High Availability Data Replication (HDR), Shared Disk Secondary (SDS), and Remote Secondary Standby (RSS) servers. Enterprise Replication (ER) provides a means of selectively replicating data between systems in near real time. The Informix Flexible Grid eliminates the administrative complexity of ER. Flexible Grid provides the ability to automatically create database objects, such as tables, indexes, and stored procedures, on all nodes within the grid as a single operation. These enhanced Enterprise Replication features provide solutions for those customers requiring reliable and quick dissemination of data across a global organization.

There is also enhanced capability for customers requiring High Availability Disaster Recovery, in the form of the ability to resend primary server indexes to secondary servers without requiring a rebuild of the index on the primary server.

Capabilities such as these enable faster, easier, and more reliable distribution and high availability of data, resulting in improved access and use throughout the enterprise.

SG24-7937-00 ISBN 0738435694