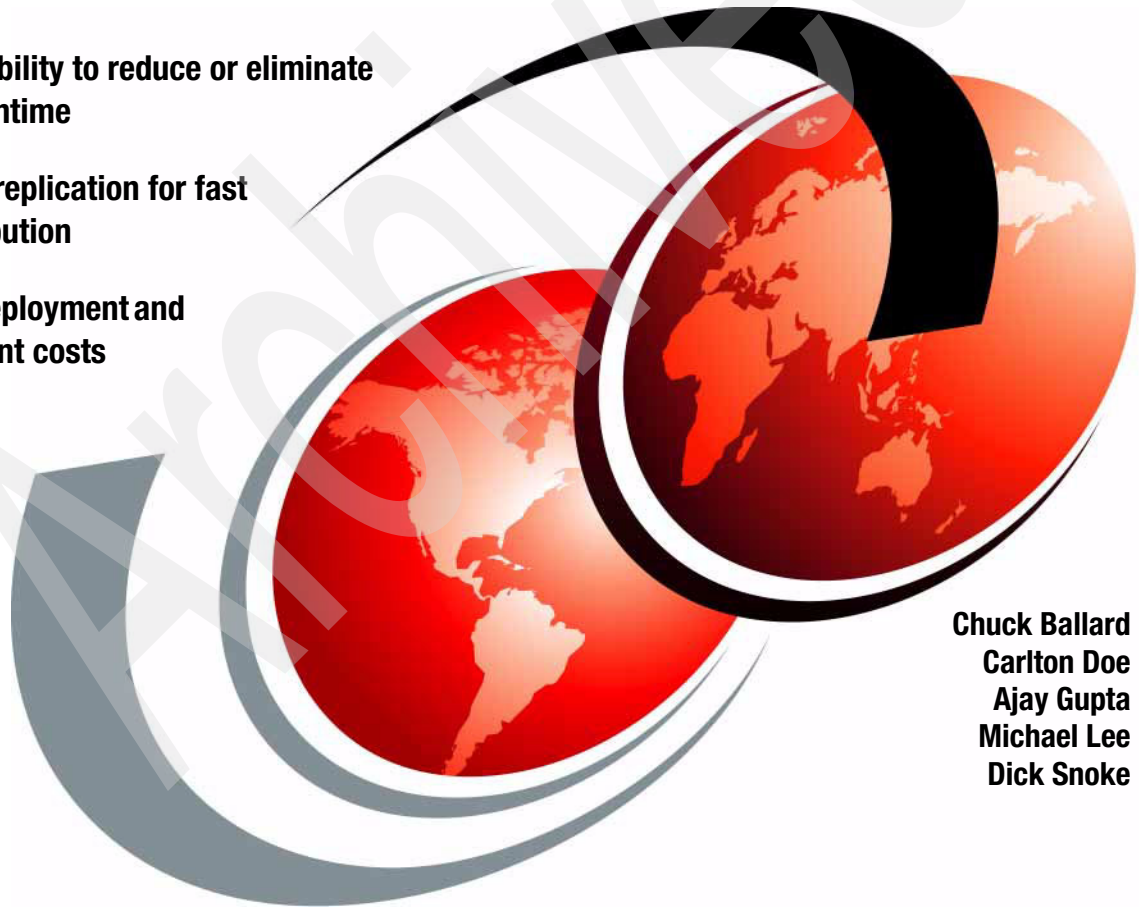


Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution

High availability to reduce or eliminate costly downtime

Enterprise replication for fast data distribution

Reduced deployment and management costs



Chuck Ballard
Carlton Doe
Ajay Gupta
Michael Lee
Dick Snoke



International Technical Support Organization

**Informix Dynamic Server V10: Superior Data
Replication for High Availability and Distribution**

April 2007

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (April 2007)

This edition applies to Version 10 of the IBM Informix Dynamic Server (IDS), and IDS 10.0.UC5.

© Copyright International Business Machines Corporation 2007. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this book	ix
Become a published author	xi
Comments welcome	xii
Chapter 1. Introduction	1
1.1 Getting started	2
1.2 HA features in IDS	3
1.3 Choosing a replication strategy	5
Chapter 2. Preparing the technical infrastructure	7
2.1 Hardware setup and considerations	8
2.2 Database server setup	9
2.3 Database server configuration	11
2.4 Database setup and considerations	14
2.5 HDR up and running	16
Chapter 3. Understanding and using High Availability Data Replication	19
3.1 What HDR is and how it works	20
3.1.1 Now or later? (asynchronous or synchronous mode)	21
3.2 Preparing for and instantiating an HDR pair	23
3.3 HDR failure recovery	27
3.3.1 The DRAUTO parameter and its impact	28
3.3.2 Step-wise approaches to recovering from an HDR failure	29
3.4 HDR and applications	33
Chapter 4. HDR: Advanced use scenarios	37
4.1 Hardware cluster for high availability	38
4.2 Combining HDR and other technology	40
4.2.1 Continuous log restore	40
4.2.2 Combining HDR with continuous log restore	42
4.2.3 Combining HDR with a hardware cluster	43
4.3 Managing multiple instances with HDR	43
4.3.1 Multiple independent instances of IDS	44
4.3.2 Star connection for multiple HDR	44
4.3.3 Cascading multiple HDR pairs	45

4.4 Data partitioning with multiple HDR	46
4.4.1 Simple table-level partitioning	47
4.4.2 Complex data partitioning with triggers and views	50
4.5 High availability solutions	53
Chapter 5. Defining the enterprise replication topology	55
5.1 Basic concepts and terms	56
5.1.1 Database servers	56
5.1.2 Replicates and replicate sets	57
5.1.3 Purposes of replication	57
5.1.4 Types of replication	58
5.1.5 Topologies for replication	59
5.1.6 Timing of replication	61
5.1.7 Including HDR in an ER scheme	62
5.2 Choosing an appropriate topology	62
5.2.1 Communications network considerations	62
5.2.2 Workload considerations	63
5.2.3 Updating and conflict resolution	63
5.3 Building a replication network	64
Chapter 6. Instantiating enterprise replication	65
6.1 The examples	66
6.2 Preparing the databases	67
6.2.1 The data and database	68
6.2.2 The IDS instances	69
6.3 Defining the ER elements	71
6.3.1 Defining servers	71
6.3.2 Defining replicates using the cdr command	75
6.3.3 Defining replicates with the Informix Server Administrator	78
6.3.4 Defining replicate sets	81
6.4 Starting enterprise replication	82
6.5 Templates	83
6.5.1 Terms and conditions	83
6.5.2 Defining a template	84
6.5.3 Realizing templates	85
6.6 Other replication considerations	89
6.6.1 Blocking replication in a transaction	89
6.6.2 Replicating user-defined types and routines	89
Chapter 7. Making changes in enterprise replication	91
7.1 Scenarios for making changes to existing ER	92
7.2 Changing a replication server	92
7.3 Changing replicates	97
7.4 Changing replicate sets	101

7.5 Template changes	106
Chapter 8. Monitoring and troubleshooting ER	109
8.1 How enterprise replication works	110
8.2 Handling asynchronous communication of ER	112
8.3 Using onstat for ER statistics collection	113
8.3.1 Global Catalog statistics (onstat -g cat)	114
8.3.2 Log snooping statistics	116
8.3.3 Grouper statistics	119
8.3.4 Reliable Queue Manager (RQM) statistics	121
8.3.5 Network Interface (NIF) statistics	128
8.3.6 Receive Manager statistics	130
8.3.7 Data Sync statistics	133
8.3.8 A few more troubleshooting tips	134
Glossary	137
Abbreviations and acronyms	141
Related publications	145
IBM Redbooks	145
Other publications	145
How to get Redbooks	145
Help from IBM	145
Index	147

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo)  ®	DB2 Universal Database™	Redbooks®
developerWorks®	DB2®	RETAIN®
AIX®	DRDA®	System p™
DataBlade™	Informix®	System x™
Distributed Relational Database	IBM®	
Architecture™	IMS™	

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

EJB, Java, JDBC, JDK, JRE, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Intel, Itanium, Pentium, Xeon, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In this IBM® Redbooks® publication we provide an overview of the high availability and enterprise replication features of Informix® Dynamic Server (IDS) Version 10. These capabilities can simplify and automate enterprise database deployment. Version 10 offers patent-pending technology that virtually eliminates downtime and automates many of the tasks associated with deploying mission-critical enterprise systems.

The high availability capability can reduce or eliminate costly downtime through, as examples, the introduction of online index rebuild, the capability to create and drop indexes online without locking tables, and point-in-time table-level restore. Enhanced enterprise replication provides solutions for those customers requiring reliable and quick dissemination of data across a global organization.

The replicated data can also participate in the high availability environment with capabilities such as seamless online resynchronization of enterprise replication nodes at startup, for example. There is also enhanced capability for high-availability disaster recovery customers 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.

These capabilities enable fast, easy, and reliable distribution and high availability of data, enabling improved access and use throughout the enterprise.

The team that 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. The team members are depicted below, along with a short biographical sketch of each:

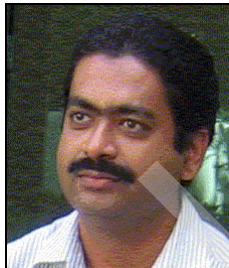


Chuck Ballard is a Project Manager at the International Technical Support Organization, in San Jose, California. He has over 35 years of IT experience, holding positions in the areas of product engineering, sales, marketing, technical support, and management. His expertise is in the areas of database technologies, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and

seminars worldwide. Chuck has both a Bachelors degree and a Masters degree in Industrial Engineering from Purdue University.



Carlton Doe has over 10 years of Informix experience as a DBA, Database Server Administrator and 4GL Developer before joining Informix in 2000. During this time, he was actively involved in the local Informix user group and was one of the five founders of the International Informix Users Group (IIUG). Carlton has served as IIUG President, Advocacy Director, and sat on the IIUG Board of Directors for several years. He is best known for having written two Informix Press books on administering the IDS database server as well as several IBM whitepapers and technical articles. Carlton currently works for IBM as a Senior Certified Consulting IT Specialist in the Global Technical Partner Organization. He lives in Dallas, Texas.



Ajay Gupta is the Software Architect for Backup/Restore, High Availability and Data Replication Technology. He has held several technical leadership positions in the Development and QA organizations working on all major components of the IBM Informix Dynamic Server. Before joining IBM, he worked in the Operating Systems Group of the National Center for Software Technology (NCST), a premier R&D organization in India. Ajay holds a Master of Technology (M.Tech) Degree in Computer Science from the Indian Institute of Technology (IIT), Bombay, India



Michael Lee is based out of the San Francisco Bay Area. He started his career at IBM, but could not pass up an offer in 1987 to work at the company about which he had written his senior paper: Informix Software. Mike has held a number of positions, all focused around the database industry. He has experience serving as an IT applications programmer, Value Added Reseller, Instructor, Manager, Systems Engineer, and, currently, as a Certified Consulting IT Specialist architecting solutions for customers using Informix, DB2®, and Information Integration products. He holds a BS degree in Business Administration, Information Systems Management from California State University, Hayward.



Dick Snoke is a Senior Certified IT Specialist in the ChannelWorks group in the United States. Dick has 33 years of experience in the software industry. That experience includes activities such as managing, developing, and selling operating systems and DBMS software for mainframes, minicomputers, and personal computers. His current focus is on the IBM DBMS products and, in particular, the Informix database products. Dick also supports related areas, such as information integration and high availability solutions.

A special thanks to:

Nagaraju Inturi for his contributions to this book. He has been working on Informix Dynamic Server Enterprise Replication for the past 8 years. Nagaraju is an IBM Certified Solutions Expert and an Informix Dynamic Server Systems Administrator.



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

From IBM locations worldwide

Cindy Fung - Software Engineer, IDS Product Management, Menlo Park, CA.

Pat Moffatt - Program Manager, Education Planning and Development,
Markham, ON Canada.

Madison Pruet - Software Engineer, R&D Informix Dynamic Server, Dallas, TX.

From the International Technical Support Organization

Mary Comianos - Operations and Communications

Deanna Polm - Residency Administration

Emma Jacobs - Graphics

Alison Chandler - Editor

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks document dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

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

ibm.com/redbooks

- Send your comments in an e-mail 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

Introduction

Over the past ten years or so, it can be argued that society in general has adopted more of an “I want it now” type of mentality. While sociologists, psychologists, psychiatrists, anthropologists, and even moralists and ethicists discuss and debate the scope, merits, impacts, advantages, and disadvantages of this shift, from a computing perspective there is no ambiguity; all components of systems managed are to be up and available constantly with little, if any, downtime for maintenance and repair.

From a database server perspective, Informix Dynamic Server (IDS) has always led the market in features and functionality that provided high availability (HA) and enabled businesses to provide the “constantly on” computing resources that many now consider to be a requirement. Of the two major components of functionality most commonly cited, High Availability Data Replication (HDR) and Enterprise Replication (ER), only one was designed for an HA role. The design and implementation goals of the other were and are quite different, being automated data distribution within the computing environment.

This chapter briefly introduces the concepts of high availability and data replication, discusses the similarities and differences between the two and touches on some of the HA-oriented features that are provided by IDS. The remainder of the book provides hands-on guidance and direction on how to use HDR and ER technologies.

1.1 Getting started

To put the features and technologies of HA and ER into the proper context, we begin with some definitions and terminology.

For our purposes in this book, the term or concept of computing High Availability (HA) refers to the online capability or availability of components in a computer system. This availability covers both hardware (such as CPU, memory, disk, and backplane) and software (for example, operating system, utilities, and applications such as database servers) components. HA is *not* the same as Continuous Availability (CA), which is defined for the purposes here as a system that is always on or rarely ever is unavailable due to maintenance or other reasons. It is also not the same as a Fault Tolerant system, which uses near-line or online duplicate hardware and, to a degree, software to survive a component failure, resulting in a very highly available, and possibly a continuously available system.

HA technology can be used to build a CA system. But to do so means you must identify and militate each Single Point of Failure (SPoF) in the system. However, the costs of doing so escalate almost exponentially as more single points of failure are identified and failure tolerances decrease. For example, it is easy and relatively inexpensive to militate against disk failure through the use of RAID technology. Costs begin to increase when a SPoF such as disk-to-server connections, backplane, and even electrical power are considered. For example, while many disk enclosure manufacturers provide duplicate or failover power supplies, these only address a failure of that power supply and do not protect against a general power outage. There are many other things to consider in that situation. As examples, even if an uninterruptible power supply (UPS) is purchased there must be redundant circuits to provide power in and out of the UPS. Then if a breaker fails, another is already online and can carry the load. Is there service to the building from different locations so if a truck knocks over a power pole the building does not lose service? Is there a generator system to provide backup power in case of an extended outage? How much extra fuel is there and how would it be loaded into the generator? What procedures are in place to secure and deliver additional fuel if the on-hand supply runs out? All these questions and costs are just to provide HA for disks! Each component of a computing system adds another layer of complexity and cost as its SPoFs are identified.

Another brief example concerns network access. Are there redundant Internet connections that use different providers and technologies? The network connections would hardly be considered robust if both providers used the same Fibre Optic conduit to reach the building. Nor would it be wise to use providers that in turn draw from the same router one level above them. Ideally the

connections would use different physical transport layers (copper and fiber?), arrive at the building from different paths, use different upstream connections to the backbone providers, and so on. Building this level of reliability requires a great deal of time and money, but illustrates what is involved when trying to create HA components to build an FT system that can become more CA.

HA components utilize Failover (FO) technology, or the (usually automated and automatic) transfer of service and processing support from a failed component to a similar, available component. For some components, such as disks, this might involve mirroring, where disk operations are duplicated on two separate sets of disks. *Replication* is also used to provide FO from a transaction perspective. In this case, logical data operations are transferred with a guarantee of delivery and application from the originating server to one or more servers so the transaction results can be used from those servers as well. This is not the same as *copying* data or transactions because with a simple copy, transaction state as well as conflict resolution inherent with replication is not involved. It is possible to copy an older transaction over a newer one, but not with replication.

As mentioned in the chapter introduction, most administrators would also include High Availability Data Replication (HDR) as well as Enterprise Replication (ER) in the IDS HA portfolio. They would only be half right in that they both replicate data. There are significant differences in design goals and implementation of the technologies which makes HDR another part of the IDS HA portfolio, but not ER.

1.2 HA features in IDS

Informix Dynamic Server has a number of HA components that help it survive and continue operations even though there might be hardware or software failures. With its innovative and unmatched light-weight threaded Dynamic Scalable Architecture (DSA), most database server operations are isolated from all others so if one fails, it does not affect the rest of the server¹. This is leveraged to a significant degree with DataBlade™ and User-Defined Routines (UDRs). The functions and access methods can, and usually are, executed by a specific Virtual Processor (VP), so restricted access rights can be enforced as well as isolating their actions from core database server functionality.

Long before RAID became the ubiquitous technology it is today, IDS provided the ability to mirror chunks from within the database server. Not only did this provide protection against disk-related failures, it also enabled a large performance boost. Since the database server is aware of both the primary and

¹ For more information about the IDS Dynamic Scalable Architecture, see *Informix Dynamic Server V10... Extended Functionality for Modern Businesses*, Ballard et al, <http://w3.itso.ibm.com/abstracts/sg247299.html?Open>

mirror disk devices, the SQL optimizer uses both for satisfying I/O operations, directing writes to the primary side while non-conflicting read operations are directed to the mirror copy. With this, I/O throughput could be maximized depending on the ratio of read versus write operations. This technology still exists in the server today, though it is used less and less frequently.

Another significant HA component is the ability to tune and modify almost all the instance parameters or database objects while it is on line and processing end-user requests. In some cases, thanks to DSA, the database server will automatically adjust instance resources such as memory pools for an unexpected heavier than normal workload. For the others, the administrative commands can be executed and either take effect immediately or gradually as older sessions disconnect from the instance and newer sessions connect. With each new release of the database server, more and more administrative functionality is either automatically managed based on administrator-defined priorities and policies, or can be changed without interrupting user operations.

Another not insignificant HA component is the IDS ability to create, and in some cases restore from, a system backup while user operations are occurring. While this is discussed in much greater detail in the previously referenced book, IDS can create a hot backup either in serial or parallel modes depending on the backup utility used, the number and types of backup devices available, and the needs of the business. This backup does *not* require splitting off a copy of the data, which could potentially impact the failover capacity of the system. From a restore perspective, the value of the HA component is that almost all storage spaces can be restored while the database instance is on line processing transactions. Naturally, user operations within the spaces being restored are blocked. But once the restore is complete, access is restored. IDS also provides several ways to back up from one system and restore to another. In some cases, these can be concurrent operations with the restore processing as the backup is being created. If this is not necessary, other options include restoring selected data sets to one or more servers running other operating systems, or full as well as partial restores to servers running the same operating systems.

As mentioned in the chapter introduction, most users would also include two other technologies in the IDS arsenal of HA components: High Availability Data Replication (HDR) and Enterprise Replication (ER). In this, they would only be half right; while both replicate data to at least one other server, the design goals and practical implementations of the two technologies are vastly different.

HDR was designed and is used to provide transaction processing failover capabilities between two physical servers. HDR leverages transparent data replication technology to replicate the results of transactions from the original, or primary, instance to the mirror, or secondary, copy without any application changes and very little if any application impact. This technology has been

available for over 15 years and is well tested and reliable. Some of the most critical public service applications in the United States are built using IDS and rely on HDR to provide uninterrupted service. It has performed nearly flawlessly even in some of the worst natural and man-made disasters of the past several years. Thousands of other customers use it as part of their business continuity plan to protect against everything from component failure to power or network outages as well as natural or man-made events.

ER, while it also replicates data, was designed and is implemented to share data objects throughout the entire database server environment. Unlike HDR, ER replicates transactions after they have completed rather than as a part of the commit/rollback. ER was designed and is implemented to provide two major benefits:

1. Improve client application performance through localized database server connections.
2. Improve availability of data for all applications. If the local server is unavailable, a replicated copy of the data can be accessed through a remote server.

The benefits HDR and ER provide do not come without a cost. Both replication technologies require additional hardware resources including servers, disk storage, network infrastructure, and so on. Additional monitoring and alarming is also required in the rare case some sort of a replication event occurs requiring administrative interaction.

1.3 Choosing a replication strategy

So which technology should a business use? The answer is this often heard, and favorite of the authors – *it depends!* Much like choosing the correct database server for the task, the correct replication technology must be used to satisfy the business need.

If an exact duplicate of the database environment, maintained in real time is required, then HDR is the only choice. If the real time requirement is lifted so that near real time is acceptable, or the need to militate against transaction waits exists, ER can be used as well as HDR in an asynchronous mode, as explained in more detail in Chapter 3, “Understanding and using High Availability Data Replication” on page 19. In deciding between the two, asynchronous HDR is easier to set up and maintain. However, it (as well as sync HDR) is limited to one copy.

If more than one server must be replicated to, only ER provides this functionality in the current version of IDS.

Perhaps the need is to replicate a subset of database tables, or maybe just a small number of columns or even columns whose values are between a certain range; ER replicate definitions allow for a very precise definition of what gets replicated. In addition, within the ER replicate definition, specific target instances can be specified rather than all available instances. It is possible then to define some replicates to send data to one set of targets while other replicates within the same instance send data to other target instances.

What is wonderful about IDS is that these technologies are not mutually exclusive. For any given instance, IDS can support HDR for HA concerns as well as ER to distribute or receive data as needed.

Preparing the technical infrastructure

In this chapter we describe the setup of a High Availability Data Replication (HDR) environment using Informix Dynamic Server V10. HDR is very simple to configure once the requirements are satisfied. To assist you in making sure all the requirements are met, refer to the Informix Dynamic Server Administrator's Guide: Using High Availability Data Replication, G251-2267-02. It is located at:

<http://publib.boulder.ibm.com/epubs/pdf/25122672.pdf>

Here we detail how the setup requirements of HDR were met, and how we configured the book test systems to implement HDR.

2.1 Hardware setup and considerations

There were three computers available for the HDR/ER setup. The three computers, called Banda, Kodiak, and Nile, enabled us to configure an HDR pair as well as to configure enterprise replication (ER) to multiple leaf nodes. For more information about setting up ER, refer to Chapter 6, “Instantiating enterprise replication” on page 65.

According to the IDS Administrator's Guide the HDR requirements must include the following:

1. The computers must be from the same vendor, and have the identical architecture and operating system.
2. The hardware must support networking.
3. The dbspace disk naming and allocation must be identical.

The Banda computer was implemented with AIX® 5.3, and is the primary database server. The Kodiak computer operating system was AIX 5.2, and it was the secondary database server. As you can see, based on the requirements previously listed, the operating systems for Banda and Kodiak are not truly identical. However, the configuration does work properly. Future releases of the documentation in the Administrator's Guide will be revised to better clarify the requirement. That is, the O/S versions must be identical but the release levels can be different. This will work properly, and was validated in the our test environment.

HDR cannot be implemented across different architectures (32-bit and 64-bit). And so, since AIX 5.2 supports both 32-bit and 64-bit architectures, we had to confirm that both Banda and Kodiak were the same architectures. On AIX, confirmation of whether the architecture is 32- or 64-bit can be attained with the command:

```
bootinfo -K
```

On Linux®, the `uname -m` command can be used to determine 32- or 64-bit mode.

Other operating system architecture information can be found in other operating environments as follows:

- SUN Solaris™:

Command: `% isainfo -kv`

One possible reply: 64-bit sparcv9 kernel modules

- HP-UX 11.00 and later:

Command: `$ getconf KERNEL_BITS`

One possible reply: 64

► Linux:

The CPU make and model will determine whether there is 32- or 64-bit support. At the time of this writing, the AMD Opteron and Athlon 64 and the Intel® Itanium®, Xeon®, and Pentium® 4 chips support 64-bit.

Linux was the first operating system to support 64-bit processing. Check Linux vendor Web sites for 64-bit compatibility.

The Nile computer was configured using Linux (SLES 9) and functioned as a leaf node for ER. The requirements for ER are more flexible than for HDR. For example, ER supports replication from AIX to Linux. However, to create an HDR pair to replicate Nile, another Linux (SLES 9) computer would have to be configured. Our test environment is depicted in Figure 2-1.

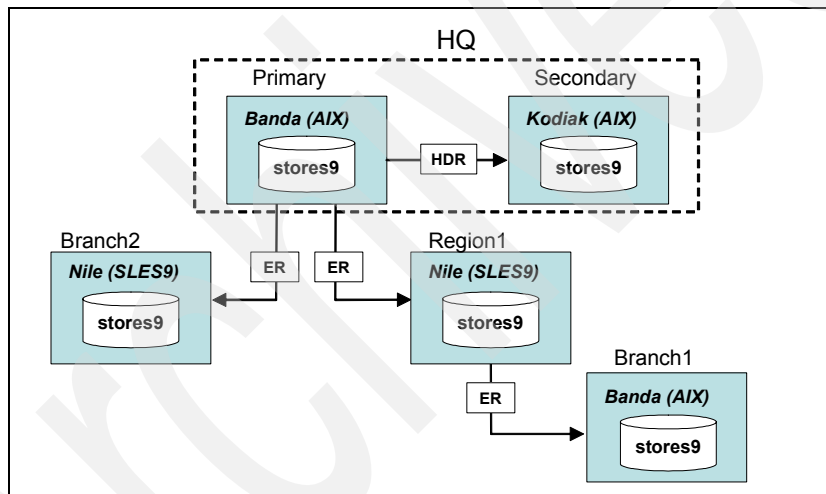


Figure 2-1 HDR/ER environment

2.2 Database server setup

For our lab configuration, the latest available release, IDS 10.0.UC5, was used. The IDS V10 installer is now Java™-based and uses a simple character-based interface. The defaults were used to get the database software installed and to quickly get the database instance installed and running.

During the install, there is the option to create a demo instance called demo_on. The default for the option is no, but we chose yes to quickly ensure that the software installed correctly and worked properly.

The install of the software on Banda and Nile was straightforward, and no problems or errors were encountered. The onconfig file, which contains all the INFORMIXDIR, INFORMIXSERVER, INFORMIXSQLHOSTS, and ONCONFIG parameters, was installed next. Once this was complete, the server was started and an **onstat** - command was executed and confirmed that the software was installed and working properly.

The install of Kodiak appeared to go as smoothly, but an error occurred during the creation of the demo_on instance. The installer did not provide any details on what error occurred, nor did the online.log file. An **onstat** command was again executed; it returned an error as shown in Example 2-1.

Example 2-1 onstat command execution

```
Kodiak: > onstat -

exec(): 0509-036 Cannot load program onstat because of the following errors:
        0509-130 Symbol resolution failed for /usr/lib/libc_r.a[aio_64.o]
because:
        0509-136 Symbol kaio_rdwrr64 (number 0) is not exported from
                dependent module /unix.
        0509-136 Symbol listio64 (number 1) is not exported from
                dependent module /unix.
        0509-136 Symbol acancel64 (number 2) is not exported from
                dependent module /unix.
        0509-136 Symbol iosuspend64 (number 3) is not exported from
                dependent module /unix.
        0509-136 Symbol aio_nwait (number 4) is not exported from
                dependent module /unix.
        0509-150 Dependent module libc_r.a(aio_64.o) could not be loaded.
        0509-026 System error: Cannot run a file that does not have a valid
format.
        0509-192 Examine .loader section symbols with the
                'dump -Tv' command.
```

The release notes (\$INFORMIXDIR/release/<language>/<lang code>) for IDS V10 were checked to make sure that all necessary OS patches were downloaded and installed. But even after applying a known patch the same error was still received. Checking again, this time in the file ids_machine_notes_10.00.txt, the solution was readily available.

Tip: This problem could also have been quickly resolved by calling IBM Informix support.

The problem was that AIX utilizes kernel asynchronous I/O (kaio), which requires a device to be created. To do that, turn on asynchronous I/O and perform the following commands:

```
su root
/usr/sbin/chdev -l aio0 -a autoconfig=available
/usr/sbin/mkdev -l aio0
```

After executing these commands, we again tried to initialize a database instance on Kodiak, using the command **oninit -ivy**. This time it worked perfectly.

2.3 Database server configuration

HDR has some requirements at the database configuration level, and they are documented in the Administrator's Guide. In our lab system configuration, all the defaults were selected that come with the `onconfig.std` file. By using the defaults, we expected no problems in getting HDR started for the first time.

Note: We discovered that even using all the defaults, a few parameters need to be considered and checked to ensure that HDR works properly.

As examples, the following `ONCONFIG` parameters need to be identical on both database servers in the HDR pair:

- ▶ `ROOTNAME`
- ▶ `ROOTOFFSET`
- ▶ `ROOTPATH`
- ▶ `ROOTSIZE`
- ▶ `PHYSDBS`
- ▶ `PHYSFILE`
- ▶ `LOGFILES`
- ▶ `LOGSIZE`
- ▶ `DYNAMIC_LOGS`
- ▶ `DRAUTO`
- ▶ `DRINTERVAL`
- ▶ `DRTIMEOUT`

Next, `dbspaces` and `chunks` need to be exactly the same down to the number of `dbspaces` and `chunks`, the pathnames, the size, and offsets.

Many users appreciate the fact that the HDR secondary is not only a hot failover, but is also a read-only database for reporting purposes. Therefore, the secondary can be configured slightly different to handle report type queries. For example, altering some of the database configuration parameters, such as BUFFERS and CPUVP, can aid reporting performance.

For even more improvement in performance, the logical and physical logs were moved out to their own dbspaces. Moving the logical and physical logs to their own dbspaces is a common performance tuning practice and is highly recommended. If at all possible, put the logical and physical logs on separate controllers from the database data.

Determining the correct size of the logical log not only can affect the overall performance of the database, but it also affects HDR.

As seen in Figure 2-2, when the logical log buffer is flushed from memory to disk an entry is put into an HDR buffer, which is part of the virtual shared memory. It is important to note that the HDR buffer is allocated based on the size of the logical log. Therefore, it is important to give close attention to the size of the logical log buffer and how many logical logs are configured. Creating a logical log buffer that is too large could affect the checkpoint duration on both the primary and secondary. Coupling the large logical log with buffered database logging could also impact performance. Tips on estimating the size of the logical log buffer can be found in the IBM Informix Dynamic Server Administrator's Guide.

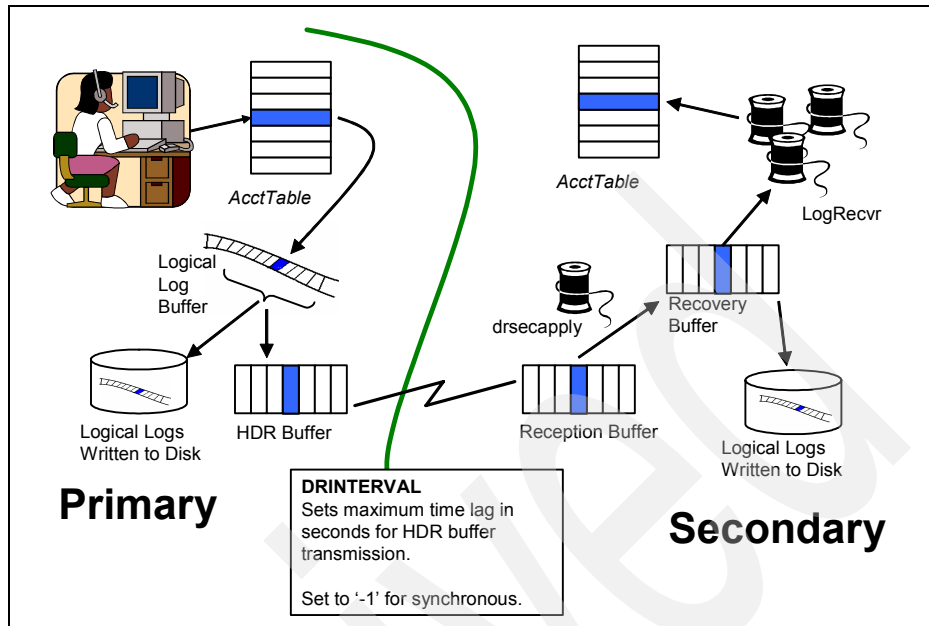


Figure 2-2 How HDR works

The only database configuration parameters that affect HDR are DRTIMEOUT, DRINTERVAL, and OFF_RECVRY_THREADS. Here is a brief description of each:

- ▶ **DRINTERVAL** configuration parameter: Indicates whether the data replication buffer is flushed synchronously or asynchronously to the secondary database server. In years past, slower networks experienced some network bandwidth problem when the DRINTERVAL was set to zero. However, with the high network bandwidths available today, and the small (2K) HDR TCP packets, network traffic is not an issue.
- ▶ **DRTIMEOUT** configuration parameter: Determines how long to wait for a transfer acknowledgement from the secondary server. This parameter usually does not require much change from the default of 30 seconds. However, problems can occur if the primary does not receive an acknowledgement from the secondary and then starts putting transaction information to the file identified in the DRLOSTFOUND database parameter. The secondary will switch to the mode specified in the DRAUTO data configuration parameter (RETAIN®, OFF, or REVERSE_TYPE). See the Informix Administrator's Guide for more details on DRAUTO.
- ▶ **OFF_RECVRY_THREADS** database parameter: A key parameter for the secondary server that controls the number of recovery threads that are run in

parallel to apply the changes that are coming to the secondary server to be applied.

For additional information about this topic, refer to sections 3.1.1, “Now or later? (asynchronous or synchronous mode)” on page 21 and 3.3.1, “The DRAUTO parameter and its impact” on page 28.

2.4 Database setup and considerations

Now that all the hardware and database server requirements have been met, it is time to focus on the database requirements at both database servers.

Transaction logging (buffered or unbuffered) must be turned on at both databases to allow the changes from the primary to be applied to the secondary. One way to check whether or not logging has been turned on is by using the **onmonitor** command. While logged in as Informix, issue the command **onmonitor** then choose **L** (for Logical Logs), and then **D** (for Databases). The resulting screen will list the databases within the instance of the INFORMIXSERVER for which the environment variable is set, and show the logging status. The following is the legend for terms used:

- N - No logging
- U - Unbuffered
- B - Buffered

Logging can be changed by using the following three utilities: **ontape**, **ondblog** and **onbar**. **Ontape** and **ondblog** examples are provided in Example 2-2.

Example 2-2 Database logging

1) **ontape**:

```
ontape -s -L 0 [ -B <database name> ] [ -U <database name> ]
```

where :

- L 0 = level zero backup
- B = buffer logging
- U = unbuffered logging

2) **ondblog**:

```
ondblog <new mode> [-f <filename>] [<database list>]
```

new mode:

- buf - Change database to buffered mode.
 - unbuf - Change database to unbuffered mode.
 - nolog - Change database to no logging. (not in EDS)
 - ansi - Change database to be ANSI-compliant
 - cancel - Cancel logging request.
- f <filename> File with list of databases for logging change.

<database list> List of databases for logging change.

3) onmonitor (Unix only):

Choose menu: Logical-Logs

Choose menu: Databases

Look for the Log Status column for the following values:

U - unbuffered

B - buffered

N - No logging

A - ANSI

Move cursor using arrow keys to the database to be changed logging and press Ctrl-B. Then choose desired logging type.

Note: These commands tell the database server that the logging has changed, but the change will not be permanently set until a level 0 backup, using `ontape` or `onbar`, has been executed.

Assuming that both the primary and secondary databases were setup using logging, we used an already open window to the secondary server to test the database using `dbaccess`. We unexpectedly encountered a -329 error. Using `onmonitor`, we found that database logging was enabled on the secondary, but that database logging had not been turned on at the primary.

To resolve this problem, we simply changed the logging on the primary database using the `ontape` utility. This utility changed the logging, but also allowed us to take a backup of the primary. There are various ways to take a backup using `ontape`. Refer to the IBM Informix Backup and Restore Guide, G251-2269, for additional details.

For our exercise, we backed up the primary using the path of least resistance, which was backing up to disk using the `ontape` utility. We then transferred the backup file to the secondary and restored the secondary. To do this, first create a file on the file system:

```
touch /backup/stores_level0
```

Next, execute the command

```
ontape -s -L 0 > /backup/stores_level0
```

The backup file was then copied to the Kodiak server and restored to the database using `ontape -p` and declining the backup of logical logs and archive level 1.

A new option for `ontape` in IDS V10 is to change the `TAPEDEV` variable to `STDIO`. One benefit of using `STDIO` is that a named pipe can be created and

used. Named pipes provide a buffer mechanism to save on storage space and execution time. Therefore, instead of writing to disk, using FTP to transfer the file to the secondary and then using the transferred file to restore the secondary, we could have used ontape to write to the named pipe and had the secondary run a restore and receive the data through the pipe – thus never having to stage the data anywhere.

Using TAPEDEV = STDIO also allows for a faster method of restoring a level-0 to the secondary for the initial HDR setup. Then ontape and onbar can use an external backup and restore (EBR) method. Using an EBR can dramatically reduce the time needed to create a secondary machine in an HDR pair if the database is very large. Refer to the IBM Informix Backup and Restore Guide, G251-2269, for more information about External Backup and Restore.

2.5 HDR up and running

Once logging has been turned on and HDR has been started, check to see if HDR is indeed running between the two servers. To quickly check to see if HDR is up and running, use **onstat** - and look for the following output:

- Primary: IBM Informix Dynamic Server Version 10.00.UC5 -- On-Line (Prim) - Up 30.00.41 - 168072 Kbytes
- Secondary: IBM Informix Dynamic Server Version 10.00.UC5 -- Read-Only (Sec) - Up 30.00.41 - 177072 Kbytes

You can also check for HDR specific threads using the **onstat -g ath** command as shown in Example 2-3.

Example 2-3 HDR specific threads

Banda: > onstat -g ath

Primary

15	4094e920	4021b954	2	cond wait	drcb_bqf	1cpu	dr_prsend
16	40982bf0	4021be60	2	sleeping	secs: 13	1cpu	dr_prping
17	40982eb8	4021c36c	2	sleeping	secs: 3	1cpu	dr_idx_send

Kodiak: > onstat -g ath

Secondary

17	4099b418	4021c36c	3	cond wait	netnorm	1cpu	dr_secrvcv
18	409a0018	4021c878	3	cond wait	drcb_bqf	1cpu	dr_secapply
19	409a0640	4021cd84	3	sleeping	secs: 25	1cpu	dr_secping
20	409a0908	4021d290	3	sleeping	secs: 4	1cpu	dr_idx_recv

Another method of checking the status of HDR is to look in the online.log file (onstat -m).

Summary of steps preparing for HDR

The objective of this chapter was to provide a brief overview of how to prepare database servers for HDR. As mentioned many times, the IBM Informix Administrator's Guide is very thorough, giving step by step instructions on getting HDR up and running.

The following is a summarization of those steps:

1. Check that all hardware requirements are met.
2. Check that all database server requirements are met.
3. Check that all database requirements are met.
4. Take a level-0 archive of the primary.
5. Execute the command on the primary:
`onmode -d primary <secondary- instance name>`
6. Restore the level-0 backup to the secondary.
7. Execute the command on the secondary:
`onmode -d secondary <primary instance-name>`
8. Execute the command on the secondary:
`ontape -l`
9. Check the message logs on both machines (onstat -m) to verify that the machines are connected and communicating.

Congratulations, you are finished.

Understanding and using High Availability Data Replication

Informix Dynamic Server, since its earliest versions, has always tried to provide as much uninterrupted access time as possible. For example, most administration tasks can be executed while the instance is handling end-user operations and backup, and some restore operations can occur under a full user load. IDS contains a number of technologies to help the instance withstand logical and most physical errors. These technologies include the logical and physical logs, consistency checks, and disk mirroring. As good as these features are, if the physical server fails the instance and data access is interrupted, which will have an impact on business operations.

This is where the IDS High Availability Data Replication (HDR) functionality comes into play. Through HDR, an exact copy of logged databases within an instance are created and maintained in real time on another physical server. If the first (or primary) server fails, applications just need to reconnect to the copy (or secondary instance) to continue operations.

In this chapter we discuss and describe HDR, how to configure and instantiate an HDR pair, failure recovery and application design and connectivity issues.

3.1 What HDR is 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 to both sets. While HDR requires two sets of storage, each is managed independently by a separate physical server. When data definition language (DDL) and data manipulation language (DML) statements are executed on the primary server, those operations are replicated on the secondary server at a predefined interval. Since most people use HDR for disaster tolerance, the interval is usually set to *immediate*. But it can be set to any reasonable positive number of seconds.

In setting up HDR, an almost-true mirror of the primary system is created. It is not a full mirror because data stored in BLOB spaces is not replicated. This 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.

Since 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 sized storage devices. The devices must be created and available at the same PATH on the secondary as on the primary. This is most easily accomplished using symbolic links as the PATH parameters in the *onspaces* creation statements.

Two other stringently enforced requirements are that both servers must be using the same O/S and IDS version. While there can be a little variation in the O/S level (for example, having different release levels of the same O/S version such as AIX 5.2 and AIX 5.3), it is not possible to set up HDR between two different operating systems or between two different versions of the same operating system. The IDS version match check is rigidly enforced. The version must be identical all the way to the sub-version identifier, such as UC2.

It is helpful, though not necessarily required, that the computing hardware be identical as well. This is particularly important if the secondary is intended to provide full user support while the primary is unavailable. While HDR will fail over to the secondary instance even if it has only 25% as much configured shared memory and CPU VPs as the primary, in the event that 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 (reports and so forth) can be executed against the secondary instance, thus reducing the load on the primary server/instance. The benefit is that 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.

3.1.1 Now or later? (asynchronous or synchronous mode)

As mentioned previously, HDR can be configured to work in synchronous or asynchronous mode. This does not affect *how* HDR works, just *when* a certain operation occurs.

Conceptually, HDR is simple, elegant, and easy to understand. When database changes occur in a logged database, the changes are recorded in the instance logical logs. HDR takes these logical log records as they are created on the primary and transfers them to a replication buffer on the secondary instance for application there. When properly instantiated, the secondary instance is in a type of permanent roll forward mode as though it was in fast recovery and applying its own logical log records. With HDR, however, it is the primary instance log records being applied.

The transfer and application of logical log records to the secondary can occur either synchronously or asynchronously, thus the *when* component of HDR.

Synchronous mode

This is the most secure mode from a disaster tolerance perspective and is controlled by the DRINTERVAL \$ONCONFIG parameter. When set to -1, whenever a logical log record is written to the HDR and logical log buffers, the log entry is sent to the secondary instance and entered into its replication buffer. The transaction is not considered completed (or rolled back) until the primary receives an acknowledgement from the secondary that the records have been received into its replication buffers. This does not mean the changes have been applied or been written to disk by the secondary, just that they have been received and are in stable, protected storage for later application on the secondary. The net result is that there are no transactions that are uncommitted or partially committed on the secondary when compared to the primary; it is a

logically consistent copy and fully capable of seamlessly replacing the primary instance in the event of failure.

This mode can have a minor impact on application performance depending on the network interconnect and distance between the primary and secondary servers. As just mentioned, from an application perspective, it will not receive a commit or rollback message from the database instance until the transaction change records are received by the secondary. This means transaction speeds can be constrained by network conditions such as speed and distance between the servers. The delay can be measured by the amount of time it takes for a network round trip or how long it takes a packet to get from the primary server to the secondary and back.

Do not think that because network speed will have an impact on HDR the servers have to be placed next to each other. Consider the amount of replication traffic that will occur as well as the speed of the network itself. One of the authors has successfully instantiated an HDR environment over a very slow partial leased line from London to Central Scotland. While network speed was *substantially* less than 1-baseT, the HDR environment worked well because there was not a tremendous amount of data to replicate once the secondary instance was created. With higher transaction volumes, network speed and distance become more critical. Generally speaking, though, with better network speeds the two HDR servers can be almost anywhere in the world and function adequately in synchronous mode.

A commonly asked question with this mode is what happens if the secondary goes away, either due to a network outage or server failure: will the primary's transaction halt until secondary acknowledgement is received? The answer to this is a qualified *yes* and *no*. The DRTIMEOUT \$ONCONFIG parameter specifies how long either server will wait to hear from the other before declaring a replication failure has occurred. From the primary's perspective, this time could start based on the transfer of a transaction when busy or just the HDR heartbeat when idle. From the secondary's perspective, it is measured by the heartbeat. What actually happens is that the servers will attempt to contact each other four times (in case there is a temporary network or server interruption) before declaring a failure. As a result, the number of seconds entered for DRTIMEOUT should be 25% of the total time an administrator feels is appropriate before declaring a failure.

Getting back to the earlier question of what happens to the primary if the secondary is not available, "yes" the transaction will hold until either an acknowledgement is received or DRTIMEOUT x 4 seconds has elapsed, at which point an HDR failure will be declared. Once a failure has been declared, the primary will not wait for the secondary but will process transactions as though

it is not part of an HDR replication pair. What happens when the secondary reconnects is discussed in 3.3, “HDR failure recovery” on page 27.

Asynchronous mode

When DRINTERVAL is set from 0 to any positive integer value, HDR operates in asynchronous mode, transferring transaction information from the primary to the secondary when either of two triggering events occur:

- ▶ The number of seconds specified by DRINTERVAL has passed.
- ▶ The HDR replication buffer becomes full.

Setting DRINTERVAL to 0 is *not* the same as synchronous mode, however. In synchronous mode, each transaction results in a send to the secondary, ensuring transaction-level logical consistency. When DRINTERVAL is set to 0, if more than one transaction completes at the same time, they will be sent as a batch. In addition, the primary does not wait for an acknowledgement from the secondary before committing the transactions back to the application.

The most significant side effect of asynchronous HDR mode is the loss of logical consistency between the instances. A transaction could be marked complete on the primary but not exist on the secondary. The larger the value of DRINTERVAL, the greater the chances of losing transactional integrity. This could have business implications if a sudden need arises to convert the secondary to standard mode in which it can process transactions.

Failure condition determination in asynchronous mode is much like in synchronous mode. The primary will attempt to communicate with the secondary four times, either through a replication buffer flush or a heartbeat, before declaring a failure. Likewise, the secondary will use its heartbeat monitor to decide if the primary is still available.

As mentioned earlier, HDR provides the greatest business value when executing in synchronous mode since a complete duplicate is being maintained. For some businesses, though, a very close approximation may be sufficient; a value of 0 or a very low number of seconds may be adequate, particularly since there is no transaction wait impact on the application.

3.2 Preparing for and instantiating an HDR pair

Instantiating an HDR pair is not difficult, but there are a couple of steps that must be rigidly adhered to in order for the process to work. The need for similar, if not identical hardware resources, particularly with respect to storage resources, has already been discussed. Likewise for the software; the IDS binaries must be identical, though there can be some minor variation with the O/S version.

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 DBSERVERNAME and DBSERVERALIASES fields. This ensures that, among other things, the replication configuration parameters such as DRAUTO (discussed in 3.3, “HDR failure recovery”) and DRINTERVAL are set identically so each instance will take the appropriate action in the event of a failure.

Once all the prerequisites have been met, the HDR initialization process can begin. It is not hard provided each step is executed in its correct order as identified in Table 3-1.

Note: The secondary server in a high availability disaster recovery (HADR) pair will *not* properly connect or initialize unless all communication with it, including HADR initialization configuration commands, occurs through its *network-based* instance alias. If tried through the instance shared memory connection, the secondary server will hang after the physical restore. After the hang an error indicating a checkpoint is required will occur, yet one cannot be forced.

If the logical restore described in Table 3-1 is attempted, it will fail even if triggered by executing the **onmode -d secondary primary_instance** command. The MSGPATH will log a message that recovery is beginning but logical log information will not be applied within the secondary instance.

Provided that all the other prerequisites have been met, and the procedures outlined in Table 3-1 are followed, using the network-based connection on the secondary server will enable a successful initialization of an HADR pair. This step is explicitly called out in Table 3-1, but is implied in all the failure recovery mechanisms discussed in 3.3, “HDR failure recovery” on page 27.

For more information, refer to the sqlhosts information in the Informix Dynamic Server Administrator's Guide, G251-2267.

Table 3-1 Steps to instantiate HDR

Primary server	Secondary server
<ul style="list-style-type: none"> ▶ Edit the HADR configuration parameters (DRAUTO, DRTIMEOUT, DRINTERVAL, DRLOSTFOUND) in the \$ONCONFIG file. ▶ Add entries to the \$INFORMIXSQLHOSTS and /etc/services files for network connections to primary and secondary instances. ▶ Modify the /etc/hosts.equiv or the .rhosts file for the informix userid to enable trusted communication with the other server. ▶ Make sure the databases to be replicated are in logged mode. ▶ Send a copy of the primary instance's \$ONCONFIG file to the secondary server. ▶ Create at least one (1) temporary dbspace if one does not exist. 	<ul style="list-style-type: none"> ▶ Add entries to the \$INFORMIXSQLHOSTS and /etc/services files for network connections to primary and secondary instances. ▶ Modify the /etc/hosts.equiv or the .rhosts file for the Informix userid to enable trusted communication.
	<ul style="list-style-type: none"> ▶ Create physical disk chunks sized identically to those on the primary server. Mirror chunks are optional. At least one (1) temporary dbspace will be required. ▶ Create symbolic links to tape devices and disk chunks pathed identically to those on the primary server. ▶ Modify the \$ONCONFIG file received from the primary server. Change DBSERVERNAME, DBSERVERALIASES, and possibly MIRROR. ▶ Install the file. ▶ Set up general Informix environment. ▶ Change the \$INFORMIXSERVER environment variable to point to the network-based instance. ▶ Initialize the secondary instance (<i>oninit -ivy</i>) and bring it all the way online to test access to all base physical devices. ▶ Using the <i>onspaces</i> utility, add all the dbspaces to make sure the devices are accessible. ▶ Monitor the MSGPATH file to make sure all the "sys" databases are built. ▶ Shut down the instance (<i>onmode -ky</i>)

Primary server	Secondary server
<p>Create a level 0 backup of the instance.</p> <p>See “Notes on the initialization process” for some important information about this step.</p>	<p>Perform a physical restore using the archive generated from the primary instance. Do <i>not</i> archive the logical logs when prompted by the command. There are no other archive levels to restore. When the physical restore completes, the instance will be in recovery mode, with all dbspaces created, but in an inconsistent state. This is normal.</p> <p>See “Notes on the initialization process” for some important information about this step.</p>
<p>With the instance in online mode, initialize the HADR mechanism in this instance and set it to primary mode by executing the onmode -d primary secondary_instance command, where secondary_instance is replaced by the network-protocol-based instance name of the secondary instance.</p> <p>The MSGPATH file should reflect this change and contain the following messages:</p> <p>DR: new type = primary, secondary server name = secondary_instance DR: Trying to connect to secondary server ... DR: Cannot connect to secondary server DR: Turned off on primary</p>	<p>After the initialization command has been executed on the primary, initialize the HADR mechanism in this instance and set it to secondary mode by executing the onmode -d secondary primary_instance command, where primary_instance is replaced by the network-protocol-based instance name of the primary instance.</p> <p>The MSGPATH file should reflect this change and contain the following messages:</p> <p>DR: new type = secondary, primary server name = primary_instance DR: Trying to connect to primary server ...</p>
<p>The primary server should show the connection completed. The MSGPATH file should contain the following message:</p> <p>DR: Primary server operational</p> <p>This can be verified by executing an onstat - command. The status line should now indicate the instance is a primary.</p>	<p>The secondary instance should be fully connected and operational. This can be verified by executing an onstat - command. The status line should reflect that the instance is acting as a secondary. MSGPATH files on both instances should indicate the beginning of logical log transfers.</p>

Notes on the initialization process

In this section we describe the process for backing up and restoring the instances.

Either the *ontape* or *OnBar* utility can be used to create the instance backup image required for HDR initialization. In the past most people have chosen to use *ontape* because of its simplicity. While *OnBar* can be used, it requires moving and configuring the bootstrap and other files to the secondary instance. In addition, only the serial *OnBar* option (**onbar -bw**) can be used, rendering the

OnBar operation identical to *ontape*. It is better to use the easier utility to begin with. With IDS V10 there is new functionality that makes using *ontape* the de facto standard for initializing HDR.

With IDS V10 a new configuration parameter has been added to *ontape*, allowing it to output to standard I/O (STDIO). With this functionality, an *ontape* backup can be piped through file handling routines such as **cpio**, **tar** or **gzip**. The *ontape* output can also be directed through a shell script that can incorporate some intelligent file handling with respect to file naming so the previous backup is not overwritten.

This flexibility also allows an administrator to do one other thing that is specifically beneficial to HDR initialization – the *ontape* backup can be redirected through a remote shell directly to the secondary instance, which can be executing a restore operation in real time with the backup! The syntax for this might look similar to:

```
ontape -s -L 0 -F | rsh secondary_server "ontape -p"
```

By executing the backup and restore in this manner, several big problems are eliminated. First, if the backup is created to tape, then shipped to the secondary's location, too much time may elapse between the backup and restore operations. The logical logs may have rolled on the primary, preventing the logical roll forward portion of the initialization. Second, depending on the size of the instance, it could take quite a while to output to tape, or require quite a few tapes to back up the primary instance. Third, having to process a backup then a restore can require quite a bit of time. With this option, the restore completes right after the backup is finished, cutting initialization time at least in half.

3.3 HDR failure recovery

Either the secondary or the primary instance can declare that an HDR failure has occurred. The process by which each decides whether or not a failure has occurred was discussed in “Synchronous mode” on page 21. Recovering from an HDR failure will vary depending on the following:

- ▶ The value of the DRAUTO configuration parameter and the amount of work (if any) performed on the former secondary instance
- ▶ If the failure was logical (or network-related)
- ▶ The amount of work processed on the primary after the failure condition was declared
- ▶ If a physical failure occurred, which server failed, and the nature and extent of the failure

In this section we discuss the HDR recovery scenarios an administrator might face.

3.3.1 The DRAUTO parameter and its impact

This parameter has had an interesting history. It was originally introduced with HDR, then removed in some of the IDS V9 releases. It has returned in Version 10 to plague unsuspecting administrators.

The intent of the parameter is a noble one – it determines what action the secondary instance should take in the event of a replication failure. In a failure the secondary, based on this setting, will do the following:

- ▶ **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 end-users, then revert back to secondary mode when the original primary returns. As part of the change back to secondary, it will transfer 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.

The problem with this parameter is that it assumes the only failure condition that will occur is an actual hardware failure on the primary that takes it out of service. While this type of failure is certainly possible, it is far more common for network-oriented issues to arise and cause a replication failure. In this case, having the secondary do anything other than remain in read-only mode will cause data integrity problems leading to the loss of some data.

When communication between the two instances is interrupted, both instances think the other has failed, and with DRAUTO set to either RETAIN or REVERSE type the secondary will switch to processing transactions. When communication is re-established with the original primary, depending on what DRAUTO is set to, the original primary may be told to convert to secondary, which it will not allow, or the secondary will try to push transactions to the original primary, which it will not receive. The only solution is to shut down the secondary and re-initialize HDR again. Any transactions processed on the secondary while it could support the full range of SQL operations will be lost, which could have a significant business impact.

For this reason, it is *strongly* recommended that DRAUTO only be set to **0** (off) unless the business can absolutely guarantee the network connections will *never* fail under any circumstances.

Handling a failover situation with DRAUTO set to 0 is not difficult and does not even require a trained IDS administrator to resolve. When a failure condition is declared, it is trapped by the ALARMPROGRAM. Depending on the business needs, anyone can be alerted to the condition. They can then log into a simple Web-based interface that asks a couple of basic *Yes / No* questions such as:

- ▶ Is the primary server offline?
- ▶ Is there a network outage between the primary and secondary instances?
- ▶ Can all users blocked from accessing the primary instance access the secondary instance?
- ▶ Will the primary instance be unavailable for longer than N minutes (where N is replaced by the maximum allowable downtime)?

Depending on what questions are asked and the resulting answers, the script running behind the interface can convert the secondary into primary mode or leave it in read-only mode if the outage will not be very long.

3.3.2 Step-wise approaches to recovering from an HDR failure

In this section, step-by-step instructions are given for recovering from a number of different failure conditions. As will become apparent, the configured value for DRAUTO has a major impact on what and how recovery occurs. By the end, it should be abundantly clear that setting it to OFF is the best choice.

Restarting HDR after scheduled maintenance

The information in this section is reasonably straightforward. As the maintenance period begins, the secondary should have been taken off line, followed by the primary. This is particularly true if DRAUTO is set to REVERSE_TYPE because the secondary will attempt to transition to primary based on the loss of the original primary.

If work needs to be done on each physical server, bring one up, do the work, then shut it down and start the other server. When starting the secondary, if the DRAUTO is set to RETAIN_TYPE, the instance will transition to primary mode but will revert back to its original state when it sees the original primary. Therefore, nothing special needs to be done to the secondary instance.

If DRAUTO is set to REVERSE_TYPE, change the secondary instance to standard mode by executing an **onmode -d standard** command before executing any maintenance work on that server. Once the maintenance work has been completed, the HDR process will need to be restarted. This is done by re-executing the **onmode -d secondary primary_instance** command where **primary_instance** is replaced with the network-based instance alias for the

primary HDR instance. Once this command is executed, bring the original primary instance on line and the two should re-connect without a problem.

Recovery following a physical failure

Recovery after a physical failure depends on which server failed, what component failed and the setting of DRAUTO. Obviously, if a non-mirrored disk supporting chunks on the secondary failed, HDR needs to be re-initialized on the secondary. The primary in this case will have declared a replication failure and have continued processing transactions. Simply execute a concurrent backup and restore operation as described previously, then execute the **onmode -d secondary primary_instance** command where **primary_instance** is replaced with the network-based instance alias for the primary HDR instance.

Primary server failure, DRAUTO is set to OFF or RETAIN_TYPE

In today's computing environment, it is almost impossible to find a server using JABOD (just a bunch of disks) storage as opposed to some RAID level or simple O/S mirroring. However, if there was a media failure or some other failure on the primary server, DRAUTO was set to RETAIN_TYPE, and a significant amount of change occurred on the secondary server during the outage (defined as the logical logs wrapping since the failure), the primary server will need to be re-initialized as follows:

1. Shut the secondary instance (acting as primary) down into quiescent mode. Make sure all transactions are closed.
2. Create a level 0 archive to disk or tape and transfer to the primary server.
3. Shut the secondary instance down.
4. Execute a physical restore on the original primary, but do *not* roll forward the logical logs.
5. Bring the primary instance on line. The MSGPATH file will indicate that a physical restore completed without a logical restore.
6. With the primary instance on line, restart HADR by executing the **onmode -d primary secondary_instance** command where **secondary_instance** is replaced with the network-based instance alias of the secondary instance.
7. Restart the secondary instance.

At this point, the two servers should re-establish HADR connectivity and continue working as before the failure.

If some other failure occurred on the primary server and DRAUTO was set to either OFF or RETAIN_TYPE, but very little change occurred (defined as all the logical logs still being available) while the secondary instance was processing transactions, bring up the primary instance and let the HADR mechanism re-synchronize the instances. If DRAUTO is set to RETAIN_TYPE, the

secondary instance automatically reverts back to secondary mode after transferring its records to the primary.

Primary server failure, DRAUTO set to REVERSE_TYPE

With DRAUTO set to REVERSE_TYPE, it must be assumed that it does not matter which server hosts the primary or secondary instances. Both physical servers are identical from a resource perspective and client applications can reach either easily. That being the case, as long as the failure was not media related and all the logical logs are still available on the new primary, bring the original primary server and instance on line. The two instances will reconnect and the new primary will transfer its logical log information to the new secondary, bringing them into logical consistency.

If the failure on the original primary was media related, or the logical logs have rolled onto the new primary, HDR must be re-initialized on the new secondary. With the original primary instance (soon to become the new secondary) shut down, execute a simultaneous backup and restore from the new primary as described previously. When completed, execute the **onmode -d secondary primary_instance** command where **primary_instance** is replaced with the network-based instance alias for the new primary HDR instance.

Should you decide that the location of the primary HDR instance *is* important even though DRAUTO is set to REVERSE_TYPE, recovery requires the execution of the following steps:

1. Shut down the new primary instance to quiescent mode. Make sure all transactions have been completed.
2. Create a level 0 backup to disk or tape. If the servers are some distance apart, two copies of the backup are required.
3. Shut down the new primary instance.
4. Re-initialize the original primary instance by executing a physical restore from the backup created from the new primary instance. Do not restore any logical logs.
5. Bring the original primary instance on line. The MSGPATH file will contain messages that the physical restore completed but the logical restore did not occur.
6. With the original primary server on line, restart HDR on this instance by executing an **onmode -d primary secondary_instance** command where **secondary_instance** is replaced with the network-based instance alias of the secondary instance.
7. Re-initialize the secondary instance back to its original state by executing a physical restore from the backup created in step 2.

8. Restart the HADR mechanism on the secondary instance by executing the **onmode -d secondary primary_instance** command where **primary_instance** is replaced with the network-based instance alias for the original primary HDR instance.

The two servers should re-establish the HADR connection and begin working as before the failure.

Recovery after a network failure

When DRAUTO is set to OFF, recovery of a secondary instance after a network failure is identical to recovery after a physical failure of the secondary. If the logical logs have rolled onto the primary, HDR must be re-initialized with a backup, restore, and re-execution of the **onmode** command.

If the logical logs are still available, a decision must be made on the acceptable length of time to bring the two instances into logical consistency. While the primary will automatically begin transferring its logical log records to the secondary, it may be faster to re-initialize the secondary with the backup/recovery/**onmode** command sequence than to wait for the logical roll forward process to execute. If the decision is made to re-initialize the secondary, shut it down prior to restoring the network connection so the logical roll forward process does not begin.

If DRAUTO is set to RETAIN or REVERSE type, consider the following questions before proceeding with a recovery process:

- ▶ What did the applications do when the failure occurred?
- ▶ If the applications reconnected to the new primary instance, how critical is it to preserve work executed on the new primary instance during the failure condition?

As we discuss in the next section, application connectivity can be configured so that applications can automatically connect to more than one instance. When this is configured, it is very important to proceed cautiously before restoring the network connection, particularly if the applications were able to connect to one or the other of the instances based on their location in the network topology. If a connection was possible, it is likely there will be transactions executed against both instances.

With DRAUTO set to either of these types, when the network connection is re-established, each instance will assert to the other that it is the primary instance and attempt to transfer its records of changes to the other; they will not be accepted.

Ultimately, the business needs to decide whether the value of the transactions on the new primary is great enough to justify either reprocessing the work on the

original primary server or extracting the changes from the new primary instance's logical logs with the *onlog utility*. While the *onlog utility* itself is easy to use, reading and understanding its output is challenging and requires a careful and thorough review of its documentation. This documentation provides a complete description of how to interpret the header and action codes in the log records. Depending on the number of tables that experienced changes, the *onlog utility* can be run with flags that extract only one table's information. It is also possible to restrict the output to just one user ID.

Regardless of the decision on data reconstruction, the secondary server will need to be re-initialized to recover from a network failure when DRAUTO is set to these types. As a result, the instance should be shut down prior to re-establishing the network connection. Once the network connection is restored, execute the backup, restore, and HDR/**onmode** commands as described previously.

3.4 HDR and applications

It is important to realize that from an application perspective there is no difference between a primary and a secondary HDR instance other than that one is read-only. If permitted by the business (because of licensing reasons), and depending on the log transfer mode used, a smart programmer will use both instances to the benefit of the application. Applications could be written such that updates and modifications occur at the primary instance while report functions or applications are directed to the secondary instance. This could be accomplished through hard coding the instance names into the application, or, preferably, changing the environment on which the application operates.

How current the data is on the secondary instance depends on DRINTERVAL. If set to asynchronous mode, the data on the secondary server will not be an exact duplicate of the primary's data at any moment in time. This may not be important if, for example, the time range for the report does not conflict with the replication interval.

So how can an application be directed to connect to one instance or another, and if HDR is used, can an application automatically reconnect to the secondary instance if the primary experiences a failure?

As should be well understood, application connectivity is determined by a series of parameters or environmental variables. These include \$INFORMIXSERVER, DBPATH, and the SQLHOSTS GROUP option, as well as the **connect** SQL statement. Of these, the SQLHOSTS GROUP option is the newest and probably the least well known.

The most common method for directing an application to an instance is through the **connect** SQL statement. While it can contain the syntax of **connect to database_name@instance_name**, it typically only holds the database name relying on \$INFORMIXSERVER to point to the correct instance. Some application languages may allow this to be treated through a variable passed in from the command line, but for the most part this statement is hard-coded into the application, requiring modification and recompiling for any change to take effect. This is hardly a seamless change from a user perspective if it must be switched during an HDR failure. For this reason, most applications only declare the database name in the statement and use INFORMIXSERVER to find the correct instance.

The INFORMIXSERVER parameter can either contain a single instance name *or* a server GROUP, as we explain shortly. In theory, if one instance is unavailable, all that is required is to reset the value of this parameter to find the new instance. Unfortunately, the value of INFORMIXSERVER is only read when the application is started, so applications must be restarted to recognize a change. This is not a seamless change from a user perspective, but it can work, and many sites use this approach.

If INFORMIXSERVER is not defined, *or the instance defined in INFORMIXSERVER is not available*, the application will look for and use the list of instances in the DBPATH environment parameter. This permits the first of two methods of more seamlessly switching application connectivity in the event of failure. For this to occur, INFORMIXSERVER is populated with the primary instance name while DBPATH is configured with the secondary instance. In the event of an HDR failure, the application will attempt to reach the INFORMIXSERVER instance. If it does not respond, the application will then attempt to connect to the DBPATH instance. With this approach, the application does not need to be recompiled or restarted. Transactions that were in-flight, however, will need to be restarted since the newly promoted primary does not have any of the shared memory buffer or transaction state information that existed on the original primary instance.

Using server groups

Created for Enterprise Replication connections, a group of one or more instances can be defined within the SQLHOSTS file. ER connectivity is based on these groups, but the groups can also be used for HDR. Figure 3-1 illustrates the concept behind a server group; Table 3-2 shows the creation of a server group.

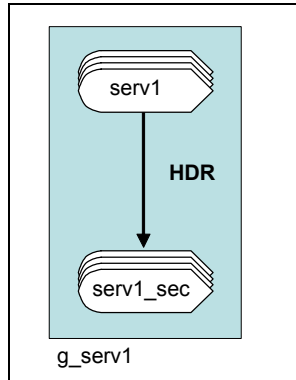


Figure 3-1 Conceptual illustration of a server group

In Figure 3-1, both the primary and secondary instances are to be configured as one logical instance named `g_serv1`. This is accomplished through entries in the client and server `SQLHOSTS` files as shown in Table 3-2. In this case, the primary instance file is shown.

Table 3-2 An `SQLHOSTS` file with server groups

DBSERVERNAME	NETTYPE	HOSTNAME	SERVICE NAME	Options
<code>g_serv1</code>	<code>group</code>	-	-	<code>i=100</code>
<code>serv1</code>	<code>onsoctcp</code>	<code>sys1</code>	<code>serv1_tcp</code>	<code>g=g_serv1</code>
<code>serv1_sec</code>	<code>onsoctcp</code>	<code>sys2</code>	<code>serv1_sec_tcp</code>	<code>g=g_serv1</code>
<code>serv1_shm</code>	<code>onipcshm</code>	<code>sys1</code>	<code>placeholder</code>	
<code>serv1sec_net</code>	<code>onsoctcp</code>	<code>sys2</code>	<code>serv1secnet</code>	

In practical terms, an application could use any of these instance definitions as its `INFORMIXSERVER` – `serv1` for read/write access, `serv1_sec` for read-only access, as well as their shared memory or regular network-based `DBSERVERALIAS`. From a failover perspective though, if `g_serv1` is used, the Informix connectivity libraries will direct an instance connection to the first instance defined in the group and if not found or unresponsive, will then attempt to connect to the next instance defined as part of the group. The application does not need to be recompiled nor restarted. Transactions that were in-flight will need to be restarted since the newly promoted primary does not have any of the shared memory buffer information.

HDR: Advanced use scenarios

There are many ways to achieve high availability today. However, it is often difficult to actually define the term *high availability* since it is a very subjective term. For example, sometimes high availability is not associated with disaster recovery, but other times the two are seen as the same. 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 resource and human problems. Disaster recovery, on the other hand, is where an external factor causes a failure and the data center needs to recover from it.

In this chapter, we discuss some of the ways to configure a system to achieve high availability with disaster recovery using technology built into IDS. Examples of these are:

- ▶ Combining HDR with continuous log restore
- ▶ Combining HDR with hardware cluster

In the current demanding environment, achieving and maintaining high availability and performance becomes a serious challenge. The information technology group is always under pressure to improve the availability of their system. On the other hand they have to manage and achieve the goal with minimum cost. These requirements result in an environment where IT manages

more than just the database servers. High availability, performance, and total cost of ownership becomes a challenging trio of demands.

In addition, there are several other factors that can add to the challenge in multiple database scenarios. For example, consider a system that was initially designed for limited capacity. As the data grows, a threshold is reached beyond which capacity planning becomes necessary. At that time, it is economical and efficient to add more database servers to handle the growing volume of data.

After mergers and acquisitions a single IT group may have to manage multiple database technologies that were initially used in those different companies. And, after the merger, it is typical that all systems are managed by a single group.

However, quite often systems are designed with multiple servers in mind. For example, data can be partitioned based on location or functionality. That is, an organization might decide to store all customer information for one geographical region (such as the Americas) on one server, and the information for another geographical region (for instance Asia Pacific) on another server. Or, if the data is partitioned functionally, all HR-related data might be kept on one server and business-related data on another server. In either situation, it may be a single IT group that will manage the data.

In this chapter we introduce some innovative techniques to manage the multiple database servers and maintain the highest level of availability. At the same time, these solutions also meet the criteria of lowest total cost of ownership.

Whether you are designing a new system from the beginning or modifying an existing schema for improvement, we also discuss what you can do to achieve a high level of availability while bringing the cost down.

4.1 Hardware cluster for high availability

HDR is great technology and many customers use it in a high availability and disaster recovery solution. But due to lack of similar technology in other software systems, HDR may not be able to provide a complete solution for high availability. For example, a customer may depend on other components of software that do not have built-in support for high availability, as does IDS, so they need to deploy other solutions that can provide high availability in their environment.

Hardware clusters also can be used for high availability. A cluster consists of two or more independent computers connected through a high-speed network, and they also share a common disk storage system. A cluster can be configured so that at any time all software is running on one computer and is accessing and

updating data on a common disk. The second computer will be a *hot* backup for the primary computer. Cluster software can be installed and running on both computers and manage both the hardware and software. The cluster software can also provide a failover capability. Many major hardware vendors sell both hardware and software to manage the cluster for high availability. A sample hardware cluster configuration is shown in Figure 4-1.

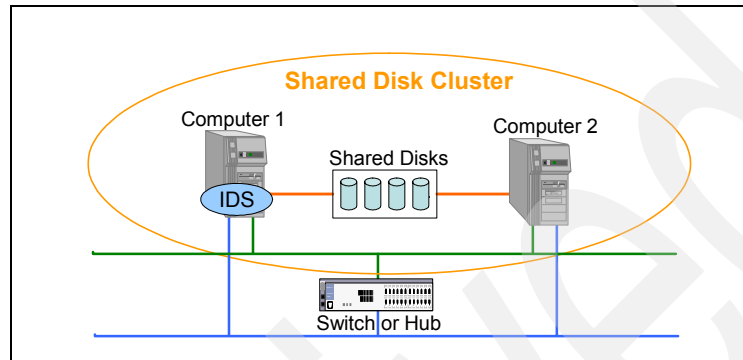


Figure 4-1 Cluster with two computer and IDS running on computer-1

In a cluster environment, IDS could be installed and configured on both nodes, using a common disk. But at any given time, the IDS server is running only on one node. In Figure 4-1, the IDS server is running on Computer 1 in the cluster. If the node (Computer 1) on which the IDS server is running fails due to any hardware or software problem, the IDS server is restarted on the Secondary node (Computer 2). This is illustrated in Figure 4-2. Starting and stopping the IDS server on any node is managed by the cluster software. The cluster software is also configured to detect failures of a node (hardware or software failures), and on a failure it migrates resources from the failed Primary node to the Secondary node, and restarts the IDS server and the applications.

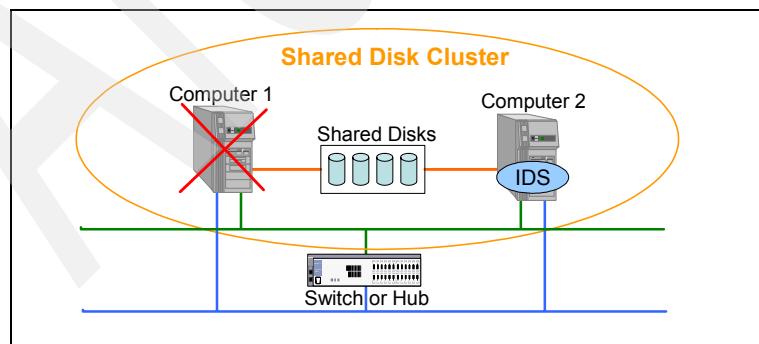


Figure 4-2 Cluster state after failure of Computer 1

Documentation on setting up IDS on an HP cluster can be found at:

http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,9928,00.html

There is, however, an exception in this document. It discusses the HP service guard on an HP mixed platform (Itanium and PA-Risc). But this combination of hardware, mixing HDR on two different platforms, is not supported.

Documentation for setting up IDS on a Solaris cluster can be found at:

<http://www.sun.com/third-party/global/db2/collateral/IDS-SC3.pdf>

This document describes the agent script to set up IDS on a Solaris cluster using Sun Cluster 3.x software. The sample agent for setting up IDS on the Solaris cluster using Sun Cluster 3.x can be downloaded from:

<http://www14.software.ibm.com/webapp/download/search.jsp?go=y&rs=ifxhaa>

4.2 Combining HDR and other technology

Today, high availability alone is not the only requirement for data centers. We have all witnessed natural and man-made calamities, and it is necessary for everyone to be ready for disaster recovery. This is not a new requirement. Traditional backup and restore does provide disaster recovery, but the traditional mechanism is not only slow, it also requires a significant amount of preparation. And there is still a concern about whether or not the system will function as planned.

Neither a hardware cluster nor HDR by itself provides both high availability and disaster recovery. However, there are many ways to configure the system to provide both. Here are two such ways:

1. Combine HDR and continuous log restore.
2. Combine hardware cluster with HDR.

4.2.1 Continuous log restore

Continuous log restore is a traditional way to set up a hot backup of a database server. The steps involved in the setup and operation for hot backup are shown in Figure 4-3. The hot backup of the Primary IDS server is maintained on the Secondary computer with similar hardware. The IDS version on the Secondary computer is also the same as that on the Primary server. A physical backup of the Primary server is made, then a backup copy is transported to the Secondary

computer. The backup is then restored (physical only) on the Secondary computer. This brings the Secondary server into a logical recovery mode. After the physical restore is complete, the IDS server on the Secondary computer is ready for a logical restore. Whenever a logical log becomes full on the Primary IDS server, it is backed up. The log backup is then transported to the Secondary computer and logical recovery (log roll forward) is performed. Using the appropriate command line option and user interaction, the hot IDS backup on the Secondary computer will suspend its log recovery. Thus the hot IDS backup is ready for a log roll forward of the next log backup.

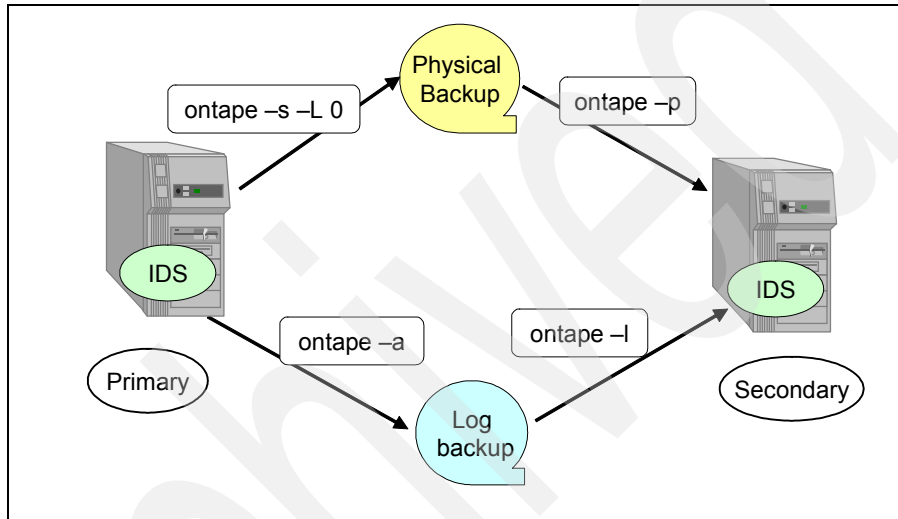


Figure 4-3 Continuous log restore

If at any time the Primary server is unavailable or fails, the Secondary server will take over. A final log recovery is performed on the Secondary server and the server is taken into quiescent mode. Later it can be brought up in online mode as the Primary server by using the `onmode` command.

In the continuous log restore mechanism, the Secondary server will always be at least one log behind the Primary server. If the last log cannot be backed up (for example, by using log salvage), all transactions in on the last log will be lost. It is also not recommended to switch to the Secondary server if the Primary server failure is temporary. If the Secondary server is brought up as Primary server, then the Primary server cannot sync up later from the Secondary server. In this scenario the Primary server needs to be set up from the current Primary server (which was the Secondary server prior to the failure). This could be a lengthy process because backup and restore is involved.

Continuous log restore is best suited for when you do not want any dependency between the primary and secondary systems. If the two system need to not be connected for security reasons, continuous log restore can still be used.

In IDS V10, you can use only ontape to perform a continuous log restore. It is achieved by suspending the log restore when all existing log backups are restored. Onbar cannot be used for continuous log restore.

4.2.2 Combining HDR with continuous log restore

A continuous log restore can be extended further to use HDR as the primary system. The concept is simple. The Primary node in continuous log restore is replaced by an HDR pair, as shown in Figure 4-4.

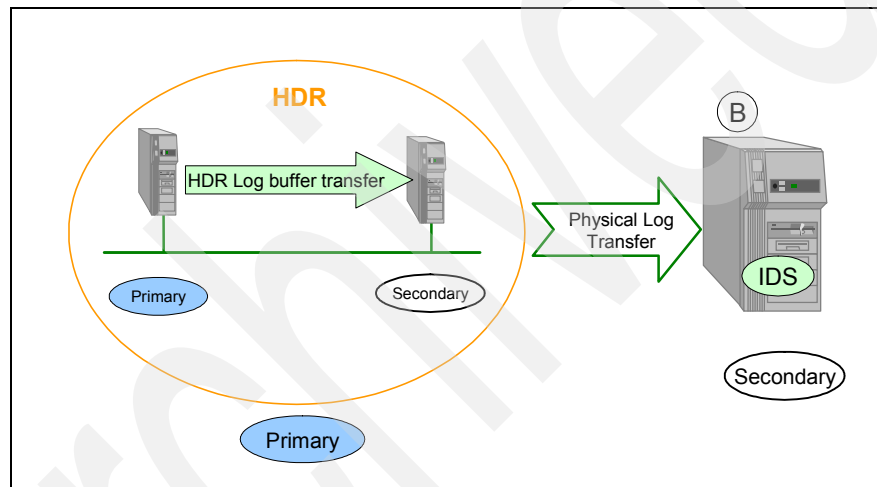


Figure 4-4 HDR with continuous log restore

The Primary node in the HDR pair will act as the Primary in the continuous log restore. A log backup is taken on the Primary server in the HDR pair and will be restored on the Secondary IDS server. The HDR pair will be configured side by side in the same data center and will provide high availability. In a typical OLTP load, the pair can be configured in sync mode to avoid any data loss. With automatic failover, availability can typically be achieved in tens of seconds. The Secondary system will be installed quite far away from Primary system. Then if there is a problem with the main data center, the Secondary system can be used by all applications.

4.2.3 Combining HDR with a hardware cluster

Another way to provide high availability with disaster recovery is to combine HDR with a hardware cluster. A hardware cluster provides a fast mechanism to support failover, but it does not provide disaster recovery. If the site fails, there is no additional backup. In addition, there is single point of disk failure. For example, a corruption in the disk may not be recoverable. On the other hand, HDR can provide reliability from many failures. But if the two systems are apart from each other, there will be some overhead required on failover. If failure is transient in nature it may not be recommended to switch over to a remote site because of that overhead. A combination of HDR with a hardware cluster can provide both high availability due to local failures and disaster recovery in situations where there is a site failure. Figure 4-5 shows an example configuration of HDR with hardware cluster.

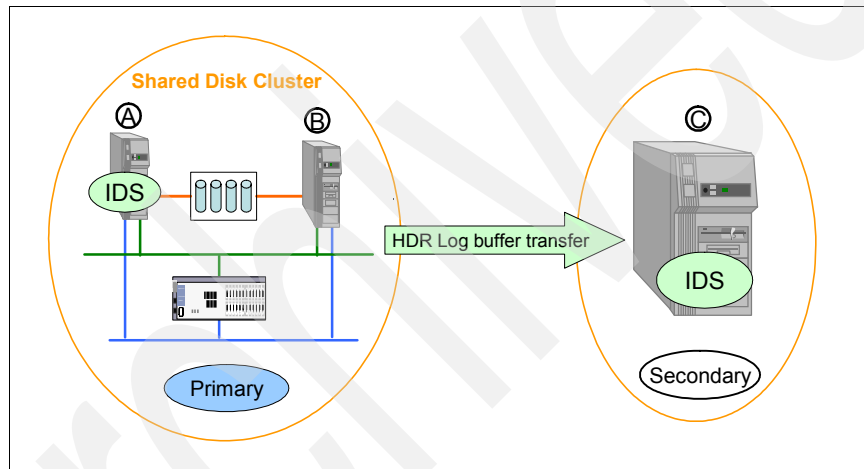


Figure 4-5 HDR and hardware cluster

This configuration requires three computers; in this example they are A, B, and C. Two of them, A and B, are part of a shared disk cluster, and IDS will be running on one of them (on A in this example). This instance of IDS will be the primary server. The third computer, C, will be used for hosting the IDS Secondary server. In this case, a failure on hardware A will be locally handled within the cluster and computer B will take over as primary.

4.3 Managing multiple instances with HDR

There are different ways to consolidate multiple instances of IDS HDR. To set up an HDR pair of single instances of IDS, two separate machines are required. In a

very simple configuration, each instance of IDS HDR can be managed on two similar machines. But this does not lead to the best possible use of hardware and may not be an efficient way of managing the system. For example, while the secondary can be used for read-only queries, that may not take full advantage of its capacity. There are two scenarios for managing multiple HDR instances:

- ▶ When all IDS instances are completely independent and service different functionality
- ▶ When multiple instances of IDS are partitioned based on data, but all of the services provided are of the same type

4.3.1 Multiple independent instances of IDS

Consider the following situation: Many of the projects in a single organization use their own database and schema, and as a result they maintain their own server instance. However, at the end of the development cycle, they are all maintained by a single IT department. All these data servers have different requirements. For example, some of them may be servicing customer requirements in different time zones, some may be used for OLTP, and some for data warehousing. When there are different types of requirements, different topologies might be developed. In the examples in this section, we consider that HA is a fundamental requirement for all data servers. We examine two common topologies here, specifically:

- ▶ Star
- ▶ Cascade

4.3.2 Star connection for multiple HDR

One of the simplest ways to provide HA is by configuring all the data servers to be on separate and independent machines. For high availability they will be configured to use HDR, and all servers will have their secondary server on a single powerful server. Such a single powerful server can be set up at a remote site. Each individual server can then reside at its own location and will be configured as the Primary server. On the remote central machine, there will be a Secondary server (IDS) for each instance of the Primary servers. This is depicted by the star topology in Figure 4-6.

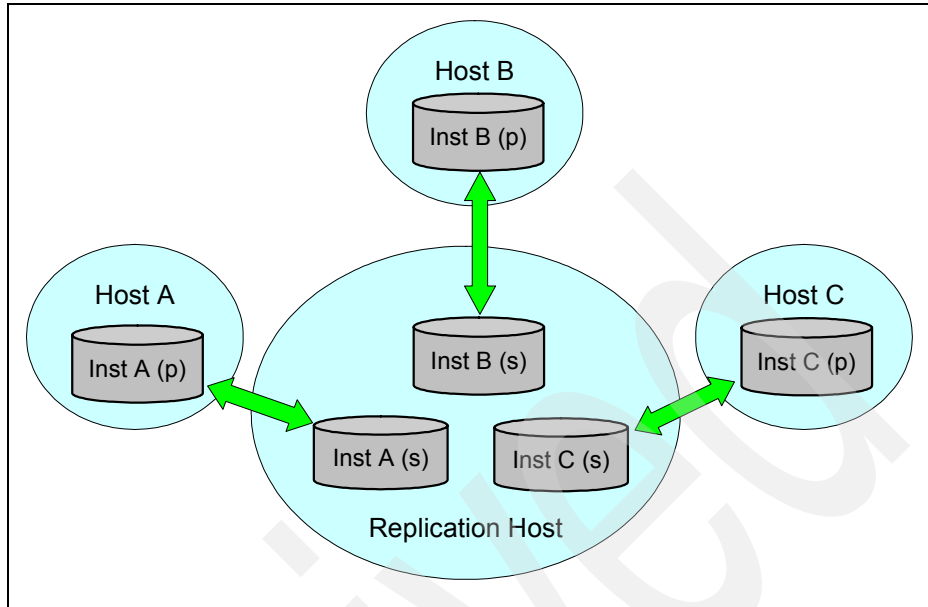


Figure 4-6 Central machine running all secondaries (star topology)

In Figure 4-6 there are three computers (Hosts A, B, and C) running individual instances of IDS (Inst A, B, and C). Each instance of IDS may be designed for individual functionality, such as HR, CRM, and Finance. They could each be located in different geographical locations based on functionality. For high availability, each of these instances will be replicated using HDR. In this case, a central computer (the Replication Host in Figure 4-6) will host 3 instances of IDS configured as Secondary servers of instances A, B, and C.

If the Primary server of any instance fails, the Secondary server on the Replication Host will take over. With a reasonable capacity, the Replication Host will be able to take the load of the failed system with very little performance degradation, if only one system fails at a time. The location of the Replication Host can be remote to handle disaster recovery.

4.3.3 Cascading multiple HDR pairs

Another way to set up multiple HDR pairs is by cascading the Primary and Secondary server, which forms a chain. Each computer will host one Primary and one Secondary system. The Secondary server of one HDR pair is configured with the Primary server of another pair, thus forming a closed chain. An example of this configuration is shown in Figure 4-7.

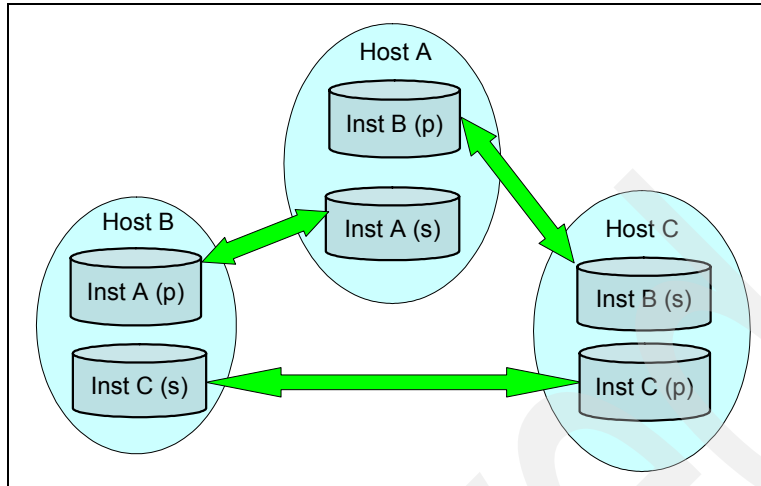


Figure 4-7 Cascade 3 HDR instance on 3 nodes

In Figure 4-7, the Primary server of instance A is configured on Host B. Its Secondary server is configured on Host A. The Primary server of Instance B is configured on Host A and its Secondary server is on Host C. Similarly, the Primary server of instance C is configured on Host C and its Secondary server is on Host A. In this configuration, the number of computers used to set up 3 HDR pairs is the same as the number of pairs. This configuration is good if you do not expect any activity on the Secondary server, or if each computer is sufficiently powerful to enable installation of two IDS servers. When one of the computers fails, the two IDS servers are left without an HDR pair. Also, the failure of one system may affect the throughput of the two IDS servers that become active on a single host.

4.4 Data partitioning with multiple HDR

High availability is becoming a necessity for all companies, and it is important to plan for it while designing a system. It is also expected that the quantity of data will increase. As a business expands or matures, the amount of data that needs to be handled grows rapidly. High availability can be coupled with a plan to handle this growth.

One of the best ways to handle data growth is by partitioning the data, which can be done in a number of ways. For example, one way to partition data is based on data type. Another way data can be partitioned is based on the locality of the data. These two strategies are discussed in the following sections.

Partitioning data basically means that you divide the data into N parts and store each part on one database server. As explained earlier, you can also use HDR with each database server to achieve high availability. Figure 4-8 shows an example of partitioning the data on two servers with HDR. Each HDR pair stores 50% of the data.

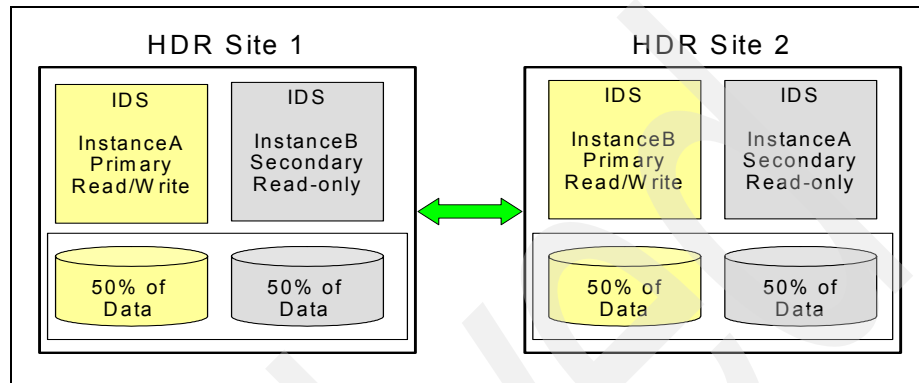


Figure 4-8 Partitioning user data into two servers

Partitioning also creates a new challenge. For example, how do you develop your application so it is transparent and does not depend on how and where it accesses the data? Many applications do take care of partitioned data. But expanding such applications becomes very difficult and also poses a high cost of maintenance. Transparency of data becomes a very important issue for application developers. In this section we describe two ways of partitioning the data. The design will take care of the fact that the application need not handle data partitioning. This will make the underlying data model on multiple servers transparent to the application.

4.4.1 Simple table-level partitioning

One of the simplest ways to think about data partitioning is based on tables. All tables in a given schema are divided into sets of tables. Each set of tables is created on a different server. Now to make all servers look uniform to the application, you need to provide access to all tables on all servers. This is where data partitioning can become an issue. One way to resolve this problem is to use synonyms. On a server, if you do not create physical tables because they belong to a set that was created on other server, then create a synonym on the local server. In this way, after schema creation is complete on all servers, the application will be able to see all tables on all servers and can access data from the physical table from any server using the local table name. The application can connect to any server to access data and need not worry about physical location of the data.

Let us look at an example. To simplify the test case, we consider a schema that consist of two tables: Customer and Item. In this example, we partition the data into two IDS servers. We first create two sets, with each set consisting of one table. The first set contains the table Customer and the second set contains the table Item. Each set of tables is created on one IDS instance (Figure 4-9).

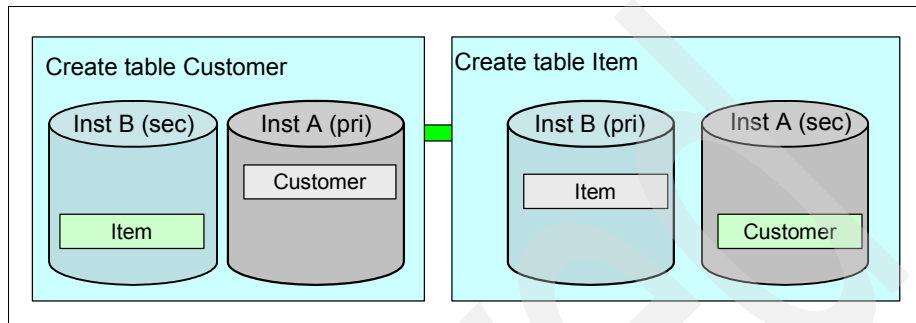


Figure 4-9 Partitioning data at the table level

Figure 4-9 shows the location of the two tables after they were created on the Primary server of two IDS instances. The Customer table is created on Instance A and the Item table is created on Instance B. Because the Customer table was created on the Primary server, it was also automatically created on the Secondary server of Instance A. The same applies to the Item table. At this time, the Item table does not exist on the Primary server or Secondary server of instance A, and the Customer table does not exist on the Primary or Secondary server of Instance B.

Create a synonym of Item on instance A using the following SQL command:

```
Create synonym Item for store@g_srvB:Item
```

Then create a synonym Customer on Instance B using the following SQL command:

```
Create synonym Customer for store@g_srvA:Customer
```

After the two synonyms are created, the two servers will appear as shown in Figure 4-10. HDR does replicate the creation of a synonym to the Secondary server. So in Figure 4-10 you can see that the synonym Item also gets created on the Secondary server of Instance A and it refers to the Item table of the Instance B Primary server. Similarly, the synonym of Customer is available on the Primary and Secondary server of Instance B. Now, the Primary server of both IDS instances A and B looks the same to an external application. Any application can connect to either of the two Primary servers and access data, run a query or perform any transactions such as updates, inserts, and deletes. The transactions

will be maintained across servers because the two-phase commit is used if transactions span across servers.

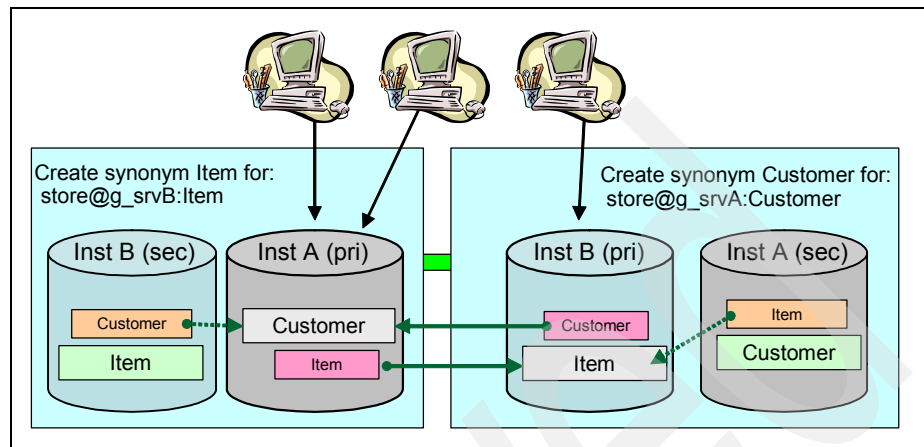


Figure 4-10 Creating views for accessing distributed data

One of the important things to notice here is that the server group is used while defining a synonym. This ensures that when you access the synonym Item on Instance A, the query will be redirected to the server which is Primary in the group.

It is interesting to see how a failover works in this scenario. Suppose, for example, that the machine where the Primary server of Instance A and the Secondary server of Instance B was running fails. At that time, the only machine that is available is the one with the Primary database server of Instance A and Secondary database server of Instance B. So, the DBA can change the role of the Secondary server of Instance B and make it the Primary server, as show in Figure 4-11 on page 50.

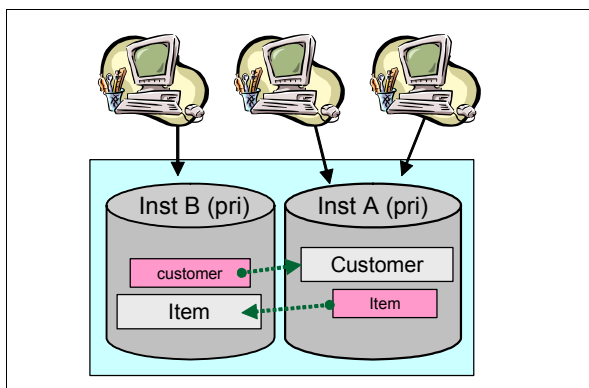


Figure 4-11 Simple data partition - Failover

If clients are connected using group, those that are accessing a database using the group name for instance B will connect to the Primary server of Instance B on the first machine. In this figure, you can see that the client can access data transparently from any of the two servers on the machine.

What is interesting is that you can extend the schema design to multiple servers with very little effort. If two servers cannot handle the load, you can add a third instance of IDS and make HDR pairs. And, you can decide then whether to use the HDR pairs in a Star or a Chain. You can also mix the schema design to a different topology, as discussed in “Managing multiple instances with HDR” on page 43.

4.4.2 Complex data partitioning with triggers and views

Partitioning data based on a table between multiple servers may not yield the best performance for some applications. It is possible that an application may be accessing data from different tables that may be partitioned across multiple servers. This could lead to quite a lot of data movement between the servers and thus to poor performance. A transaction spanning servers will internally follow the two-phase commit protocol.

Simple table partitioning does not maintain locality of data. For example, one IDS server instance will contain all data in a table. It is quite likely that the data in the table belongs to different geographical regions. If different servers are located in different geographies, it may be best to hold the active data for a particular geography on the server in that region. This leads to the notion of partitioning data based on location.

Even if the data is partitioned based on location, the application should be allowed to access all of the data from anywhere. For performance reasons, it

may be useful to contain application data specific to a region on a server in that region. But it is quite likely that an application from a different region may want to access data from any server. For example, there could be call centers for different geographies and each of them might have their own server. Then, an application from a local call center would connect to the local server and access data. During off-peak hours, only one call center might be open and all calls will be routed to that region. That means an application will be accessing all data, not data specific to a particular region. It would be good if a single application worked all the time without switching from server to server.

To achieve data transparency, but still partition the data based on locality, a complex mechanism can be used. It is assumed here that multiple IDS Instances are used for scalability as well as high availability. This can be achieved by designing the schema using views, triggers and stored procedures, which is functionality that is available in IDS. This mechanism of partitioning can be best explained using an example. So, here we extend the example from the previous section.

Assume that the Customer table will be partitioned and table data will be physically located on the server where the data logically belongs. In this case the Customer table will be partitioned based on the *cust_no* column in the table. Any customer record whose *cust_id* is less than 10000 will be stored in one server and the rest of the customer records are stored in another server. For simplicity we partition the data into two servers, creating two base tables c1 and c2 (c1 in Inst A and c2 in Inst B). As explained in the previous section, synonyms will be created on both servers to make sure that both tables are visible in each server. Figure 4-12 shows the physical location of the two tables and their synonyms on the other servers.

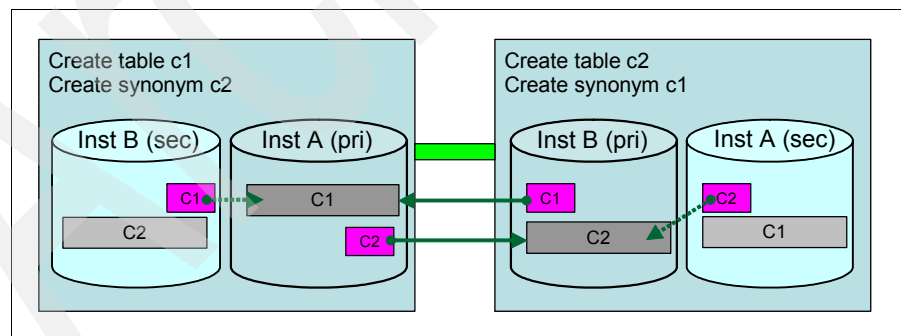


Figure 4-12 Creating two base tables c1 and c2 and their synonyms

Table c1 holds customer record with *cust_id* less than 10000 and c2 holds customer record with *cust_id* \geq 10000. This requirement has not yet been

enforced in the schema. So, now to view all customer records, create a union all view with the following commands:

```
create view customer as
select * from c1 union all select * from c2;
```

This will create a view using all of the customer table that can be accessed. Any application can use the Customer table without knowing the location of the data, which makes data transparent to the application. Figure 4-13 shows the logical and physical relationship between various database entities.

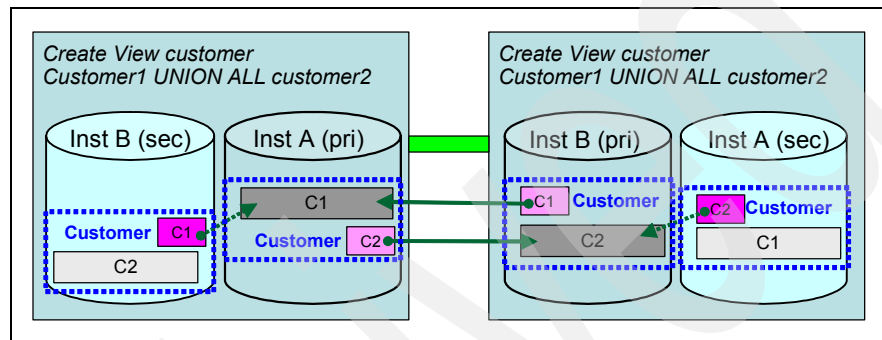


Figure 4-13 Creating partitioned data

So far, accessing data is not an issue, but what about updating? Insert, update, and delete will not work on view, so how will an application insert, update, and delete rows from the Customer view? To achieve this we use a stored procedure and a trigger. Here, the actual work is performed inside of the stored procedure. There will be trigger and stored procedure for each operation.

This is illustrated in Example 4-1, which shows a stored procedure and trigger that performs an insert into the customer table. The stored procedure is very simple, and ensures that the customer record is inserted into the correct base table. The stored procedure and trigger must be created on views on each server. This technique can be extended to perform update and delete operations as well.

Example 4-1 Stored procedure and trigger for handling inserts

```
create procedure cust_ins(a int, b char(100))
  if a < 10000
  then
    insert into cust1 values(a, b);
  else
    insert into cust2 values(a, b);
  end if
end procedure;

create trigger inst_cust_trig instead of insert on customer
referencing new as n
for each row (execute procedure
cust_ins(n.cust_no, n.cust_name));
```

Any application can treat the customer view as the customer table for all work. The application does not have to deal with location-based data and need not worry about actual location of the data. The IDS server will route the query properly and access the data in an optimized way to avoid data movement.

The beauty of this solution is that partitioning can be extended to any number of servers. It can start with single server and as the data volume grows, the number of servers can be expanded. When adding servers, the underlying schema remains transparent to the applications.

4.5 High availability solutions

We have discussed a number of solutions in this chapter, and the question might be asked, “Which is the best solution for high availability?” The answer to this question is as you might expect: It all depends. There is no single solution that will satisfy the requirements of everyone. This is because high availability requirements differ from one organization to another. Another important question to ask is, “What is the downtime expected as part of the high availability configuration?” And, you must also ask, “What cost is the organization willing to pay for the design of an HA solution?” High availability solutions not only depend on high availability of the database server, but also on the other components that comprise the ecosystem.

If application failover is a bottleneck then perhaps there should not be significant focus on achieving a very high level of availability for the database server. For example, assume that based on a fixed number of failures the application uptime is 98%. In this case even a database availability of 99.99% will not increase the

overall availability. Designing and configuring high availability is a compromise between the cost of the systems implementation and maintenance, and the loss of revenue from loss of access to the system. As the uptime requirement increases, the cost of a high availability system increases exponentially. So it is best to understand the system capabilities with failover time and the cost of a high availability solution to help determine the best strategy for your particular requirements.

Defining the enterprise replication topology

Enterprise replication, often referred to as “ER,” is a process for creating and maintaining replicas of data sources in one or more separate database instances. It is performed by a set of functions that can be combined in a number of ways to meet a wide variety of business needs. In this chapter, we introduce the major components of enterprise replication and discuss some of the ways the components can be combined. ER can be used to consolidate data from several remote systems to a central system, to distribute data from a central system to remote systems, or to share the workload by replicating changes on one system to each of the others. Combinations of these uses are also allowed, so you can meet a wide variety of requirements using replication.

In this chapter, we first define some basic terms and concepts related to ER. Then we discuss the purposes for using ER, the types of replication applicable to each purpose, and the ways in which the systems using replication can be organized. These sets of things (purposes, types, and topologies) are interrelated. There are often multiple designs that will meet a set of requirements, and there may not be a single best design for a set of requirements.

ER should not be confused with High-Availability Replication, often referred to as HDR. The two are separate facilities, but they can be used together. See Chapter 8, “Monitoring and troubleshooting ER” on page 109 for the details on doing that. This chapter focuses on ER alone.

The IBM Informix *Enterprise Replication Guide*, G251-1254, is the authoritative source for information about enterprise replication. Take the time to study that manual carefully in addition to the material presented here. The manual has all the details of the commands and allowed configurations.

Administration of ER (defining things, making changes, starting and stopping, and so forth) is discussed in Chapter 6, “Instantiating enterprise replication” on page 65. This chapter focuses on the concepts and ideas of ER.

5.1 Basic concepts and terms

The basic building blocks of any replication network are the replication servers and the replicates. Replicates can be grouped into replicate sets. The following sections describe these concepts in more detail.

5.1.1 Database servers

The basic element of any ER network is a *database server*. Any instance of IDS that is properly licensed for replication can be a database server. Do not confuse this term with the more general terms *node* and *server*. The system on which a database server executes is often called a *server*, and when networks are being discussed, both a hardware server and a database server are sometimes called *nodes*. However, in the context of enterprise replication in IDS, the basic element is an IDS instance, and that is called a *database server* in the ER manuals.

Note that this term denotes the entire IDS instance, not any specific database within the instance. An instance either is or is not an ER database server. You can choose to replicate data from only some databases within the server, but you must define the whole instance as a database server.

ER is replication between two or more database servers. Those may be on the same system (server) or on separate systems. They may be geographically near each other or far away from each other. And, they may be different IDS versions. For example, an IDS V7.31 database server can replicate with IDS V10.00 database servers. The only requirement is that there be a communications link over which the two database servers can communicate.

ER may involve many database servers organized in complex ways. This is the subject of 5.1.5, “Topologies for replication” and 5.2, “Choosing an appropriate topology”.

5.1.2 Replicates and replicate sets

Each database server doing replication knows what should and should not be replicated based on the set of *replicates* that are defined for that database server. A replicate is some set of data defined by a SELECT statement from one table. It may include all or just some columns of each row, and some or all the rows of the table. A replicate is the result set of a SELECT, and anything you can express that way can be a replicate (subject only to the constraint of being data from a single table).

The target of a replicate does not have to be the same as the source. The target may be different tables, columns, and rows from the source. It's similar to doing an INSERT INTO...SELECT FROM... statement.

This means replication is very flexible. Business requirements should drive the definitions and an ER design should include as many replicates as necessary.

In a complex replication scheme, there may be many replicates, and some of them should likely be treated as a group. That is, the set of replicates should be started, stopped, changed, and so forth, as a group. This is usually required in order to keep the data in the target database consistent when data from related tables is replicated. To make this kind of group operations easy, ER has the concept of *replicate groups*. A replicate group is just a set of replicates that are used together. They are started, stopped, and so forth, as a unit rather than as independent replicates.

With those basic ideas in mind, we can discuss how an ER scheme can be devised. One way to approach the question is to think about the purposes of replication, then the type of replication appropriate to each purpose, and then the way in which the database servers need to be organized. The combination of those results will usually provide the topology of hardware and software that you want.

5.1.3 Purposes of replication

There are several common uses for replication, such as data consolidation, data distribution, workload sharing, and workload distribution. One key to designing an ER scheme is a clear understanding of the purpose or purposes for the replication. An enterprise typically has a number of purposes for using replication, including some, perhaps, that do not fall into one of the categories described in this discussion.

Data consolidation is the replication from a number of systems to a single system. An example is having a chain of stores replicate their sales data to the

corporate headquarters. The data from the stores is collected into a single database; this is a one-way transfer.

Data distribution is just the opposite. It is the replication of data from a single source to multiple targets. An example is replicating prices from a corporate headquarters to each store in a chain. The data from the single central database is distributed to the many stores; this is also a one-way transfer.

Workload sharing is using a number of systems to work on a common database. Each system has a copy of the database, but changes are propagated to each of the other systems. That way any system can execute any transaction and the effects are reflected on all the other systems. An example is order taking using multiple call centers in different parts of the world, where each center has a local copy of the database. In this case, data is transferred from each system to each other system, and the transfers go in both directions.

Workload distribution is slightly different. 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 servers. It is not necessarily a one-way transfer, but that is often the case.

An enterprise may easily need replication for one, some, or all of these purposes. Having clear use cases will help assure that each purpose is met by a proposed replication design. A table or set of columns from a table may be needed in one or several use cases. Clear use cases helps ensure that the right replicates and replicate sets are defined. The use cases will probably also determine some or all of the replication schedule.

5.1.4 Types of replication

There are two types of replication, *primary-target* and *update-anywhere*. Primary-target is replication from one system to one or more other systems, often for the purpose of consolidation or distribution.

Update-Anywhere replication is replication of data among a set of databases where any of them may be updated. This is often the replication used for workload sharing.

An enterprise will need either or both types of replication based on its purposes for replication. The two ideas (type and purpose) go together. It is quite hard to choose a type without knowing the purpose.

5.1.5 Topologies for replication

Given the purposes, types, and use cases for a required replication scheme, a topology can be designed. There are three kinds of topologies, which can be combined in numerous ways. They are:

1. Replication topologies that are fully-connected database servers
2. A hierarchical tree of database servers
3. A forest of trees of database servers

If each database server is on a separate physical system, then the systems will be organized the same way. But if one physical system houses more than one database server, then the network of database servers and the network of physical systems may be different.

Fully-connected database servers

A fully-connected topology is one in which each database server has a connection to each other database server, as shown in Figure 5-1. In this example, there are three database servers, and each of them is connected to each of the others. Data can be transferred directly from any machine to either of the others. There is no requirement to use any intermediate system to make a connection.

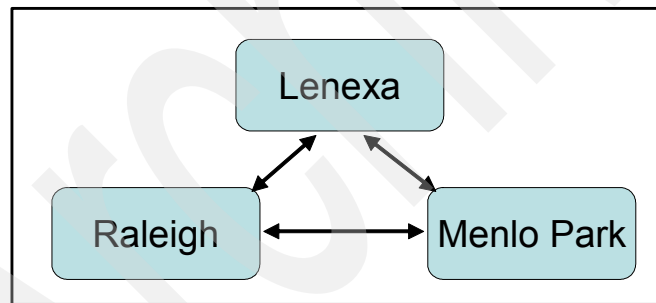


Figure 5-1 A fully-connected topology

A hierarchical tree of database servers

A hierarchical tree topology is one in which each database server is connected to exactly one other database server. An example is shown in Figure 5-2. One database server is the *root database server*, and is connected to one or more other database servers called *non-root servers*. Each of those is connected to other database servers called *leaf nodes*. Of course, there may be leaf nodes connected directly to the root node, and there may be several levels of non-root nodes. Such a tree can vary from the most simple case of one root node and one leaf node to very complex trees of dozens or even hundreds of nodes.

In this case, the Corporate HQ is the root, East Coast and West Coast are non-root servers and the cities are the leaf nodes. There might also be a leaf node connected directly to the root, for example, if the headquarters has its own branch office.

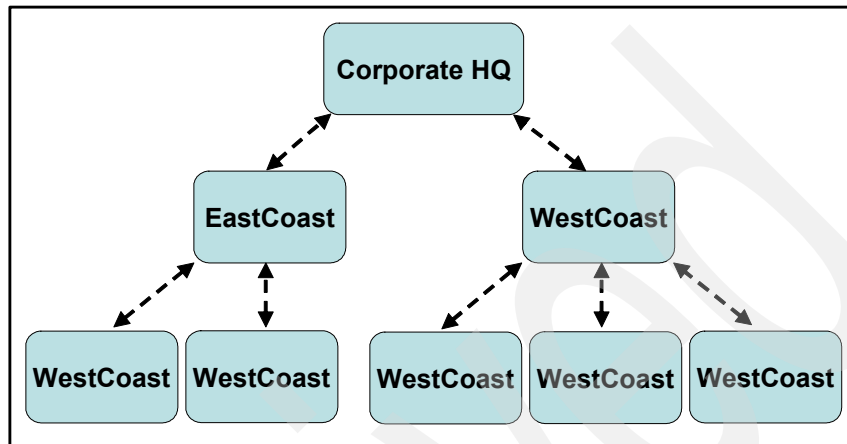


Figure 5-2 A hierarchical or tree topology

A forest of trees of database servers

A *forest of trees* topology is a set of hierarchical trees in which the root nodes of the trees are connected. Each of the trees in a forest can be different from each other tree. That means the topology can be even larger and more complex than a single hierarchical tree topology. An example of this is shown in Figure 5-3.

In this example, the US, Canada, and Latin America are peers, and each is also the root of a tree. It is easy to see that both Mexico and Brazil might have children also. There is no requirement for the trees to be balanced or to have the same number of levels.

What is important is that data from Mexico or Brazil cannot be directly transferred to any of the Canadian or US cities without first passing through the Latin America node and one other node. Network bandwidth, the capacity of each system, and the volume of data expected in various parts of the network or enterprise are important things to consider in designing a replication topology that is this complex.

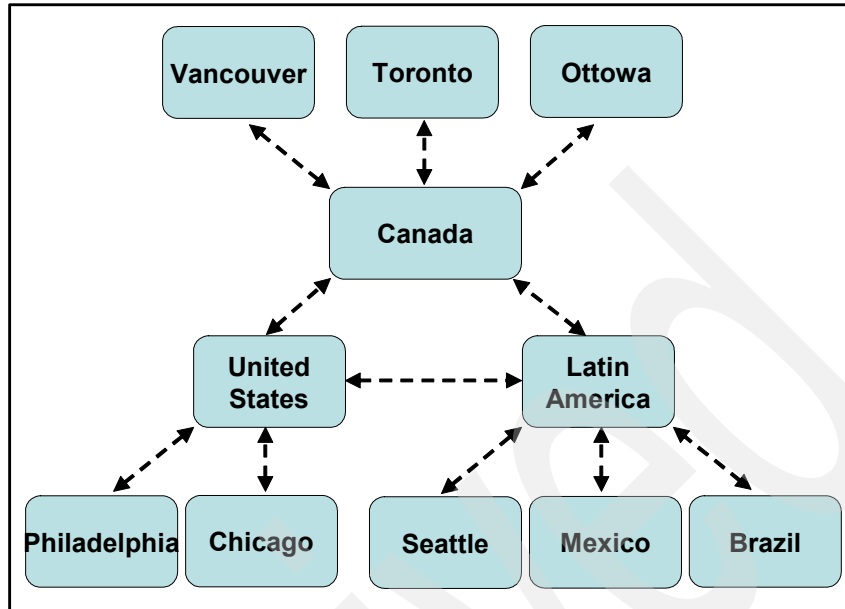


Figure 5-3 A forest of trees or multiple hierarchy topology

5.1.6 Timing of replication

Replicated data does not have to be sent between systems as soon as changes are made. That is one option, but it is not the only option. The other options are to transfer data at some regular interval or at a specific time of day.

A regular interval is a defined period of time, specified in minutes. As examples, both every 5 minutes and every 120 minutes (2 hours) are acceptable values. The data to be sent accumulates during the period between replications. So the quantity of data, the space required to hold it, and the network bandwidth required to send it all need to be considered.

A specific time of day means the data is sent at the same time every day. An example might be to replicate at 8:00 pm each evening. As with the interval, the data to be sent will accumulate until the time for replication arrives.

If either interval or specific time replication is used, the target systems do not reflect the new data or changed data until the replication is completed. Be sure your business can tolerate the delay. In addition, the changed data is held on its source database server until the specified time for the data transfer. Be sure there is enough space to hold that data. High rates of change may require more frequent replication in order to limit the space for change data.

5.1.7 Including HDR in an ER scheme

Any of the database servers in a replication network may be part of an HDR pair of instances. This might be used to ensure that key nodes, such as the root nodes in a forest of trees topology, are highly available so the flow of data is not interrupted.

In this case, only the primary system of the HDR pair is included in the replication network. The backup (or secondary) system is not directly configured for ER. An HDR pair has no greater capacity for handling enterprise replication than a single machine.

The order of creating systems does not matter. An HDR pair can be created and then added to an ER network. Or any system in an ER network can be converted to an HDR pair. What does matter is to ensure that paths are available so that the failure of any single system does not leave sets of systems cut off from one another. For example, in Figure 5-3, if one of those central servers is not available, then none of the regions are able to connect to the rest of the enterprise. Those would be good candidates to be HDR pairs.

5.2 Choosing an appropriate topology

Based on all these definitions, choosing a topology is not easy. Quite a few factors have to be studied and combined. However, there are some things to consider that may eliminate certain of the designs and reduce the number of possible solutions.

These factors can often conflict with each other. For example, the business may want an update-anywhere scheme with a fully-connected topology so that updates are propagated as quickly as possible. However, that requires the machines to have all the required network connections and the CPU power to do all the processing. In addition, maintaining the network connections may require manpower and skills beyond what is available.

The IBM Informix Enterprise Replication Guide has an extensive discussion of these topics; see that reference for complete details. Some general ideas follow, but should not be considered an exhaustive treatment of the subject.

5.2.1 Communications network considerations

The volume of data, the distance it must travel, and the network facilities that are available may dictate a hierarchical tree or forest of trees topology. For example, if replication across continents is required, then a fully-connected topology may

not be feasible because there is not sufficient bandwidth for the data volume. However, if only a few physical systems are involved, and if the data volume is not too large, such a design may be feasible. As another example, a regional retail chain may be able to fully connect the stores in the chain. But if the enterprise expands to a larger geographic area, that design may not scale. A hierarchy or forest may be required for a large number of stores.

If a fully-connected topology is chosen, then the network administration may also be a factor. Any changes to the network may require work on multiple machines. Be sure the people with the right skills are available or can be acquired before choosing this option. If not, a hierarchical scheme may be better because it isolates machines to some extent.

5.2.2 Workload considerations

If the reason for replication is that the workload has become more than its machine can perform acceptably, then replication may not be a good answer. The point is that there are costs to replication, and those costs may make replication a more expensive solution (in people, time, or money) than expanding the system.

However, if the workload can be partitioned so that there is minimal overlap, replication may work very well. For example, if an order entry workload can be partitioned so that two systems handle the work for separate geographies with little overlap, then replication is a good solution. But if each of the systems must handle all geographies, then replication costs may be larger than adding more CPUs or memory to the single system.

5.2.3 Updating and conflict resolution

If update-anywhere replication is used, then consider whether or not conflicts can occur. That is, can two transactions change the same row at the same time? If the answer is yes, then think about how to determine which of the changes will persist in the database.

If there is a deterministic algorithm to choose which of two conflicting updates is kept (and which is discarded), then update-anywhere replication may be acceptable. However, if there is not a good way to choose, then consider whether or not the conflicts can be avoided.

For example, if the application is doing order entry, new order numbers need to be generated. That will mean new rows being added to an orders table might have the same order number. It may not be possible to change the order number of either one. However, using the CDRSERIAL parameter, sequences, or some equivalent user-defined data type, the conflict may be avoidable.

Even if there is an acceptable resolution scheme for conflicting updates, be sure it covers all the cases. For example, what will happen if one transaction deletes a row that another transaction is updating? Does the row get deleted, or is the updated row retained? There are no general correct answers to such questions. Each enterprise must answer based on their specific requirements.

5.3 Building a replication network

Once a topology is chosen and the required replicates (and replicate sets) have been defined, the replication network can be constructed. That process involves a number of steps:

- ▶ Preparing 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 databases.
- ▶ Configuring the IDS instances so the resources for replication are available. Replications requires both disk space and some additional threads in the DBMS.
- ▶ Ensuring the communications connections between the nodes are correctly configured. A trusted connection between each pair of nodes that must pass replication data is required. Some additional entries in the SQLHOSTS file and some additional DBSERVERALIASes may be required.
- ▶ Producing (design, write, test, and install) any stored procedures that may be necessary for conflict resolution.
- ▶ Defining the replication servers. This identifies the systems and IDS instances that are part of the replication network.
- ▶ Defining the replicates and replicate sets. This specifies what data (tables or parts of tables) are to be replicated as well as the source and target of each replicate.
- ▶ Starting 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.

The final three steps are done using either the Informix Server Administrator (ISA) or the command line using the `cdr` command.

The best practice is to carefully record each command or activity so that the details are available for recreating the process or for debugging.

Chapter 6, “Instantiating enterprise replication” gives a detailed example of each of these steps.

Instantiating enterprise replication

In this chapter we discuss and demonstrate how to create a working replication network comprised of elements discussed in Chapter 5, “Defining the enterprise replication topology” on page 55. Chapter 8, “Monitoring and troubleshooting ER” on page 109, discusses how to adapt as the database and business evolves.

The process begins by ensuring the databases are all properly prepared for replication, and then the servers and replicates are defined. When those are all prepared, the replicates are started. That step begins the transfer of data according the replicate definitions.

An alternative technique for setting up a replication network is to use *templates*. New in IDS V10, templates simplify the administrative work of creating all the replicates on multiple systems.

We conclude this chapter with comments on replicating user-defined types and routines, and the way to execute some SQL commands without having changes replicated.

6.1 The examples

The examples in this chapter are based on a simple configuration of two machines and four instances of IDS. This configuration was chosen to demonstrate a number of features of ER without a large hardware requirement. We elected to use multiple IDS instances on each machine rather than a single instance per hardware system.

That is an important point because ER takes place between IDS instances, not systems or databases. Any or all of the databases in an instance may be the source or target of replication. Those instances may be as close as the same machine or as far apart as half way around the world. ER works equally well in both cases, and any case in between. ER can be a useful technique for sharing data among a set of IDS instances all on a single large hardware system. As with many software features, the right choice must be based on the whole set of requirements for the enterprise.

For developing this chapter, we set up four IDS instances, two instances on each of two hardware systems. One system is an IBM System p™ machine running AIX 5.3 and IDS 10.00FC5. The other system is an IBM System x™ machine running SuSe Linux Enterprise Server 9 and IDS 10.00UC5. Each instance holds a copy of the database built by the dbaccessdemo_ud program. That database is named *stores9* in each case. Two instances (HQ and branch1) also hold a second database. That is the database built by the dbaccessdemo program. This database is named *stores* in both cases. The *stores9* database is used in the examples of server, replicate, and replicate set definitions and use. The *stores* database is used in the examples of template definition and realization.

The machines, instances, and names are as listed in the sqlhosts file:

HQ	group	-	-	i=100	
ban_er1		onsocket	banda	61001	g=HQ
branch1	group	-	-	i=101	
ban_er2		onsocket	banda	61002	g=branch1
region1	group	-	-	i=300	
nil_er1		onsocket	nile	61200	g=region1
branch2	group	-	-	i=301	
nil_er2		onsocket	nile	61201	g=branch2

The hosts are named Banda and Nile.

The IDS instances on Banda are ban_er1 and ban_er2.

The IDS instances on Nile are nil_er1 and nil_er2.

For this chapter, we chose a hierarchical tree configuration. The `ban_er1` instance is the root node of the tree, and `nil_er1` is a non-root node. One leaf node is `ban_er2`, connected to `nil_er1`. The other leaf node is `nil_er2`, connected to `ban_er1`. Figure 6-1 depicts the arrangement.

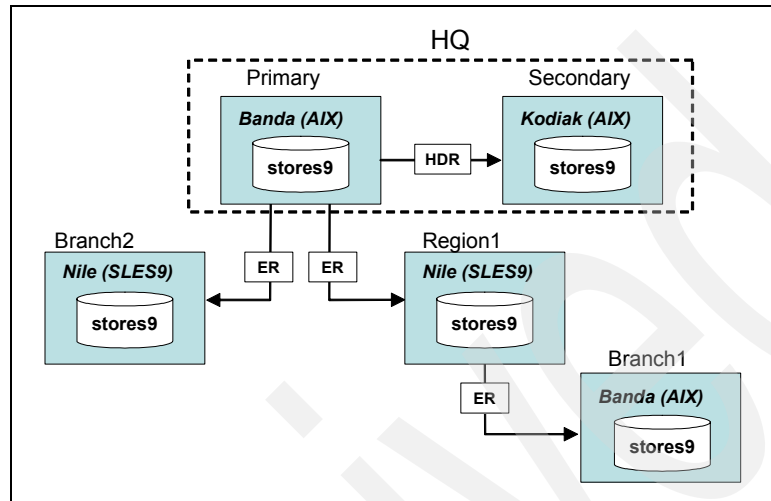


Figure 6-1 The example configuration

For demonstration purposes, the replication requirements were chosen to be:

1. Price changes (`stock.unit_price`) will be propagated to every other node.
2. Customer and sales data (`orders`, `items`, `whlsale_customer`, and `retail_customer`) will be collected at the root node.
3. Credit status (`customer.credit_status`) on either leaf node will be propagated to the root and to the other leaf node.

These are not a realistic or comprehensive set of business needs or requirements, but they are sufficient to show how to configure and use IDS replication.

6.2 Preparing the databases

A number of elements must be created and configured, and decisions made before any replication objects, such as servers and replicates, can be created.

6.2.1 The data and database

Any database used for replication must have certain properties. For example, primary keys (PKs) are required in any table that is used in replication. The PK is necessary so that the receiving DBMS can accurately identify which rows are involved in each transaction. In addition, comparison of the primary key column in each replicated change is the way conflicts are found.

Another required property is that the database must use logging. ER is not possible without logging since log records are the source of what is changed in each transaction.

If any of the tables that must be replicated does not have a primary key, the table may be altered to have an artificial primary key. This can be done in at least two ways:

- ▶ One 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.
- ▶ Another way to define an artificial primary key is to add a column. The new column might be of the serial data type or a 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 is the use of the SERIAL data type. To minimize these conflicts, the CDR_SERIAL configuration parameter can be used to change the sequences so the sequence on each machine is disjoint from the others.

To facilitate conflict resolution, two additional columns *must* be added to the tables included in replicates where conflict resolution will be defined. These two columns, often called the *shadow columns*, are **cdrserver** and **cdftime**. The two columns can be easily added to any table by altering the table to add CDRCOLS. In Example 6-1 that is done to the stock table because it is to be propagated everywhere and may, therefore, have conflicts to be resolved. The customer table may also have conflicts. However, the shadow columns cannot be directly added to typed tables, so the process for the customer table hierarchy is more complex. The data is conveniently available in load files. If that were not the case, then the tables would have to be unloaded before they are dropped. All the other tables are sent in only one direction and should not have any conflicts.

Note: Since alterations to typed tables are not allowed (error -9208), the only way to add the shadow columns is to unload, drop, and recreate with the changes and reload the tables.

```
alter table stock add CRCOLS;

DROP TABLE retail_customer;
DROP TABLE whlsale_customer;
DROP TABLE customer;

CREATE TABLE customer OF TYPE customer_t
(
    PRIMARY KEY (customer_num),
    FOREIGN KEY (customer_loc) REFERENCES location (location_id),
    CHECK (customer_type IN ('R','W')),
    CHECK (credit_status IN ('D','L','R','P','N'))
) with CRCOLS;

CREATE TABLE retail_customer OF TYPE retail_t UNDER customer with
CRCOLS;

CREATE TABLE whlsale_customer OF TYPE whlsale_t
(CHECK (terms_net >= 0 OR terms_net IS NULL))
UNDER customer with CRCOLS;

LOAD FROM 'retail_cust.unl' INSERT INTO retail_customer;

LOAD FROM 'whlsale_cust.unl' INSERT INTO whlsale_customer;
```

In the examples in this chapter, we chose not to cause any conflicts in the typed tables. That means there will be no changes to the customer.customer_num or either of the children of that table (retail_customer or whlsale_customer). In any production system, this potential conflict would have to be avoided or dealt with. One option is to use sequences instead of the serial type to generate the numbers. The sequence definition provides appropriate controls for generating unique numbers in multiple systems.

Our example systems are on two different hardware types, the IBM System p and the IBM System x. One of the example replicates is stock.unit_price. It is a floating point number, so it may have different representation on the different systems. To avoid problems, floating point numbers can be replicated in IEEE format. That is part of defining the replicates.

6.2.2 The IDS instances

The IDS instances that are replication servers must have certain resources configured so the replication threads have room to perform their functions. The

configuration parameters in the onconfig file (and the values used in the example systems) include those identified in Table 6-1.

Table 6-1 Configuration parameters

Parameter	Value	Description
CDR_EVALTHREADS	2	# evaluator threads (per-cpu-vp, additional)
CDR_DSLOCKWAIT	5	# DS lockwait timeout (seconds)
CDR_QUEUEMEM	4096	# Max amount of memory for any CDR queue (Kbytes)
CDR_NIFCOMPRESS	0	# Link level compression (-1= never, 0= none, 9= max)
CDR_SERIAL	1000,1	# Serial Column Sequence
CDR_DBSPACE	cdrspc	# dbspace for syscdr database
CDR_QHDR_DBSPACE	cdrspc	# CDR queue dbspace (default same as catalog)
CDR_QDATA_SBSPACE	cdrqueues	# List of CDR queue smart blob spaces

Each instance in the example has a unique set of values for CDR_SERIAL. That allows the use of the SERIAL type in a way that avoids conflicts. If that change to the sequence of generated numbers cannot be made for some reason, then the conflict resolution scheme *must* be able to handle the problem.

The CDR_QUEUEMEM, CDR_QHDR_DBSPACE, and CDR_QDATA_SBSPACE parameters define the space for the queue headers and data held or received by the replication threads. Each of these needs to be defined using the **onspaces** command before replication can be started. See the IBM Informix *Enterprise Replication Guide*, G251-1254, for advice on how to determine a good size for this dbspace and smart blob space.

The CDR_DBSPACE is the dbspace in which the syscdr database is created. This may be an existing space or a space that is added for this purpose. In the example, we chose to create a dbspace specifically for this database.

The CDR_EVALTHREADS defines how many additional threads will be used in each CPU virtual processor (VP) to evaluate what data from each SQL statement should be held for replications. The choice of number here depends on the volume of work. For the example, we chose to use only a single thread because the workload was very light.

Next, at least one of the DBSERVERNAME or DBSERVERALIASes must be configured in the sqlhosts file as a trusted connection to each other replication server with which it communicates. This means s=0 and s=2 cannot be used in field 5 of the sqlhost entry. In addition, no communications support modules (CSM) can be used in the sqlhosts entries.

Finally, if the replicated data should be encrypted as it is transferred between database servers, then a number of parameters must be set in the onconfig file. These include ENCRYPT_CDR, ENCRYPT_CIPHERS, ENCRYPT_MAC and ENCRYPT_SWITCH. The values of these parameters combine to determine the specific encryption applied to each data transfer.

6.3 Defining the ER elements

There are two ways to perform the administration of replication objects. One is to use the **cdr** command, which is a text-based command used from the command prompt. The other is the Informix Server Administrator (ISA), used in a browser. A few examples using ISA are included in this book, but most of the examples use the command line interface. Both interfaces offer the same functionality.

6.3.1 Defining servers

The command to start the process of defining the replication servers is **cdr define server**. Example 6-2 on page 72 shows how to set up the example servers. The root node (HQ) is defined first. Then the leaf node on Nile, which is connected directly to the root node, is defined. The third step is to define the non-root node on Nile. And then the leaf node connected to the non-root node is defined.

The command syntax has two optional styles. The example uses both, but prefers the shorter form. For example, the **-i 500** in one style would be **--idle=500** in the other style. Or the **--init** in the first command of one style in the example would be **-I** in the other style.

For the example, we chose not to use many of the options of the **cdr define server** command. Particular attention should be given to the Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) directories and files, which are in **/tmp** by default. They should usually be somewhere else, because these files contain data from rows that failed to replicate correctly. Such data should be in a directory with very limited access.

The example includes the **cdr list server** after each server definition, but that is not required. It is in the example simply to verify that the servers were properly created.

An essential point is that *the time for each of the commands to complete is unpredictable*. The last server definition in the example failed on the first attempt, but it succeeded when it was retried after a short wait. This happens because when a server is defined a number of things have to be done before the operation is fully complete. One of those things is the building of the **syscdr** database on the instance. That is similar to the work of building the **sysmaster** database when an instance is initialized. It happens in the background and some other work can proceed before it is finished. The **cdr list server** command executed correctly and showed the **region1** before that server was fully ready. After a brief delay to let the background processing complete, the failed command could be completed correctly.

Also notice the last command in the example. It shows all four replication servers, but the last one (branch1) has no status. That is because the other two servers (branch2 and region1) are connected to the root server, but the branch1 server is not. If the **cdr list server -c nil_er1** command were executed, the output would be different because that server has a different set of components connected directly.

Finally, consider the timeout value, which is 500 in the example. That is the number of minutes (8 hours and 20 minutes in this case) that an idle connection is held open if there is no traffic. You may want a shorter timeout period, but base the choice on the business requirements.

Example 6-2 Defining replication servers using the cdr command

```
banda:/home/informix $ cdr define server -c ban_er1 -i 500 --init HQ
banda:/home/informix $ cdr define server -c nil_er2 -i 500 -L -S HQ --init
branch2
banda:/home/informix $ cdr list server
SERVER          ID STATE   STATUS   QUEUE  CONNECTION CHANGED
-----
HQ              100 Active   Local    0
branch2         101 Active   Connected 0      Sep 12 11:37:53
banda:/home/informix $ cdr define server -c nil_er1 -i 500 -N -S HQ --init
region1

banda:/home/informix $ cdr list server
SERVER          ID STATE   STATUS   QUEUE  CONNECTION CHANGED
-----
HQ              100 Active   Local    0
branch2         101 Active   Connected 0      Sep 12 11:37:53
region1         300 Active   Connected 0      Sep 12 11:39:10
```

```
banda:/home/informix $ cdr define server -c ban_er2 -i 500 -L -S region1 --init
branch1
command failed -- invalid server (47)
```

```
banda:/home/informix $ cdr list server
SERVER          ID STATE   STATUS   QUEUE  CONNECTION CHANGED
-----
HQ              100 Active   Local    0
branch2         101 Active   Connected 0      Sep 12 11:37:53
region1         300 Active   Connected 0      Sep 12 11:39:10
```

```
banda:/home/informix $ cdr define server -c ban_er2 -i 500 -L -S region1 --init
branch1
```

```
banda:/home/informix $ cdr list server
SERVER          ID STATE   STATUS   QUEUE  CONNECTION CHANGED
-----
HQ              100 Active   Local    0
branch2         101 Active   Connected 0      Sep 12 11:37:53
region1         300 Active   Connected 0      Sep 12 11:39:10
branch1         301 Active           0
banda:/home/informix $
```

When a replication server is defined, the IDS server does a number of things. For example, a number of threads are started, as identified in Table 6-2.

Table 6-2 *Threads*

Number of threads	Thread name	Thread description
1	ddr_snoopy	Performs physical I/O from logical log, verifies potential replication, and sends applicable log-record entries to Enterprise Replication.
1	preDDR	Runs during queue recovery to monitor the log and sets blockout mode if the log position advances too far before replication resumes.
1	CDRGfan	Receives log entries and passes entries to evaluator thread.
n	CDRGevaln	Evaluates log entry to determine if it should be replicated (n is the number of evaluator threads specified by CDR_EVALTHREADS). This thread also performs transaction compression on the receipt of COMMIT WORK and queues completed replication messages.

Number of threads	Thread name	Thread description
1 per large transaction	CDRPager	Performs the physical IO for the temporary smart large object that holds paged transaction records. Grouper paging is activated for a transaction when its size is 10 percent of the value of SHMVIRTSIZE or CDR_QUEUEMEM or when it includes more than 100,000 records.
1	CDRCparse	Parses all SQL statements for replicate definitions.
1 per connection	CDRNsTnCDRNsAn	Sending thread for site.
1 per connection	CDRNrn	Receiving thread for site.
2...n	CDRACK_n	Accepts acknowledgments from site. At least 2, up to a maximum of the number of active connections.
# CPUs...	CDRD_n	Replays transaction on the target system (DataSync thread). At least one thread is created for each CPU virtual processor (VP). The maximum number of threads is 4*(number of CPU VPs).
1	CDRSchedMgr	Schedules internal enterprise replication events.
0 or 1	CDRM_Monitor	Monitors and adjusts DataSync performance for optimum performance (on the target).
0 or 1	CDRDTCleaner	Deletes (cleans) rows from the deleted rows shadow table when they are no longer needed.

As these threads are created, the data structures used by each thread are allocated in shared memory.

In addition to starting these threads and building the necessary data structures, an additional database, called syscdr, is built. The syscdr database is where the controls and statistics for replication are held on disk. Since this database is subject to change in each release, we do not describe it in detail here. For the full description of the syscdr database, refer to the \$INFORMIXDIR/etc/syscdr.sql file.

6.3.2 Defining replicates using the `cdr` command

Once the replication servers are defined, the details of what data should be sent to the servers can be specified. That is done in the form of replicates and replicate sets.

A *replicate* is the definition of some set of data on a server to be sent elsewhere. The `cdr define replicate` command is used to do this.

Example 6-3 on page 76 shows the replicate definitions corresponding to the requirements of the example. The `cdr define replicate` command can be quite long, so we chose to use a text file to construct the command rather than typing multiple lines each time.

The first replicate definition is very simple. It specifies the one-way propagation of changes to the order table on branch2 (one of the leaf nodes) to the HQ server (the root node). No conflict resolution is required, and all the other parameters use the default values. Best practice is to use descriptive names for the replicates so that a casual observer can understand what is defined.

Note: The group names are used, not the instance names.

This replicate does not specify anything about when or how often changed data should be transferred. The default is to move the data immediately at the end of each transaction. The alternatives are to move data at some regular interval (every 15 minutes, for example) or at some specific time (at 8:00 pm local time each evening, for example).

This replicate has two participants, denoted by the P and R (the last two lines of the replicate definition). Each participant defines the database and table that is the source (P or primary) or target (R or recipient) of the data. Data changed in the primary of a replicate is propagated to the other participants. Data changed in a recipient is not propagated. In this case, the replicate is one way because there is a single source and a single target.

Alternatively, a replicate may be defined with two or more primary participants. That is how an update-anywhere replicate is defined. In that case, some conflict resolution rule is required because the same row may be updated by separate transactions on different servers at the same time. As the data is propagated, the DBMS must be told which changes override others.

The next several replicate definitions complete the set required to have the orders, items, and customer data propagated as the requirements stipulate. Because only a single table can be used in each definition, separate replicates are required for each table whose data needs to be shared.

The last two definitions are for the last two requirements. In the *prices* replicate, there is a single source and multiple targets. Note that the destination for each replicate is declared as a SELECT statement rather than an INSERT statement. Note also that the `--floatieee` option is used so the floating point numbers are transferred correctly between the 32-bit and 64-bit systems.

The *credchg* replicate is an update-anywhere replicate. There are not Ps or Rs to denote sources or targets. Each participant in the replicate has both roles. We chose to use the time stamp as the means of resolving conflicts. To make problem resolution easier, use the `--ats` option; the `--ris` option specifies that data for which a conflict cannot be resolved should be retained so that someone can decide what to do with it.

In the first replicates, all the columns of every row are replicated. In the last two replicates, only some of the columns are replicated. If only some columns are included, the replicate must contain the primary key, even if that is not really what should be replicated. So the only way to replicate a single column is for that column to be the entire primary key.

Note that after a replicate is defined, no data is transferred. That only happens after the replicate is started. Best practice is to define all the replicates before starting any of them. That reduces the workload to synchronize the data when the replicates are started.

Notice that the `cdr list replicate` command, abbreviated in the example to `cdr list repl`, shows some of the replicates are read-only. That is not strictly correct. Those replicates are master-slave or uni-directional replicates. The read-only designates which server is the recipient of the data. In the first replicate definition, the HQ server receives the changes, but changes there are not propagated. Changes on the branch2 server are sent to the HQ server, however.

Notice that some of the replicates transfer all of the columns in the rows of a table, but others transfer only some columns. In the first case (all of the columns are replicated), you may choose to have the whole row transferred even if only some of the columns change. Or, you may choose to send only the changed data. Sending only changed data may reduce the network traffic significantly.

Example 6-3 shows the ISA interface for defining a replicate.

Example 6-3 Defining replicates

```
cdr define repl -C ignore bandaorders \  
"R stores9@HQ:informix.orders" "select * from orders" \  
"P stores9@branch2:informix.orders" "select * from orders"  
  
cdr define repl -C ignore nileorders \  

```

```

"R stores9@HQ:informix.orders" "select * from orders" \
"P stores9@branch1:informix.orders" "select * from orders"

cdr define repl -C ignore bandaitems \
"R stores9@HQ:informix.items" "select * from items" \
"P stores9@branch2:informix.items" "select * from items"

cdr define repl -C ignore nileitems \
"R stores9@HQ:informix.items" "select * from items" \
"P stores9@branch1:informix.items" "select * from items"

cdr define repl -C ignore bandar_cust \
"R stores9@HQ:informix.retail_customer" "select * from retail_customer"
\
"P stores9@branch2:informix.retail_customer" "select * from
retail_customer"

cdr define repl -C ignore niler_cust \
"R stores9@HQ:informix.retail_customer" "select * from retail_customer"
\
"P stores9@branch1:informix.retail_customer" "select * from
retail_customer"

cdr define repl -C ignore bandaw_cust \
"R stores9@HQ:informix.whsale_customer" "select * from whsale_customer"
\
"P stores9@branch2:informix.whsale_customer" "select * from
whsale_customer"

cdr define repl -C ignore nilew_cust \
"R stores9@HQ:informix.whsale_customer" "select * from whsale_customer"
\
"P stores9@branch1:informix.whsale_customer" "select * from
whsale_customer"

cdr define repl -C ignore --floatieee prices \
"P stores9@HQ:informix.stock" \
"select stock_num, manu_code, unit, unit_price from stock" \
"R stores9@region1:informix.stock" \
"select stock_num, manu_code, unit, unit_price from stock" \
"R stores9@branch1:informix.stock" \
"select stock_num, manu_code, unit, unit_price from stock" \
"R stores9@branch2:informix.stock" \
"select stock_num, manu_code, unit, unit_price from stock"

```

```
cdr define repl -C timestamp --ris credchg \  
"stores9@HQ:informix.customer" \  
"select customer_num, credit_status from customer" \  
"stores9@region1:informix.customer" \  
"select customer_num, credit_status from customer" \  
"stores9@branch1:informix.customer" \  
"select customer_num, credit_status from customer" \  
"stores9@branch2:informix.customer" \  
"select customer_num, credit_status from customer"
```

6.3.3 Defining replicates with the Informix Server Administrator

Figure 6-2 shows the first ISA window for creating a replicate. In this window the name and other properties of the replicate are provided, but the participants in the replicate are not defined. When the values in this window are filled in and the **define replicate** button is clicked, the replicate is defined.

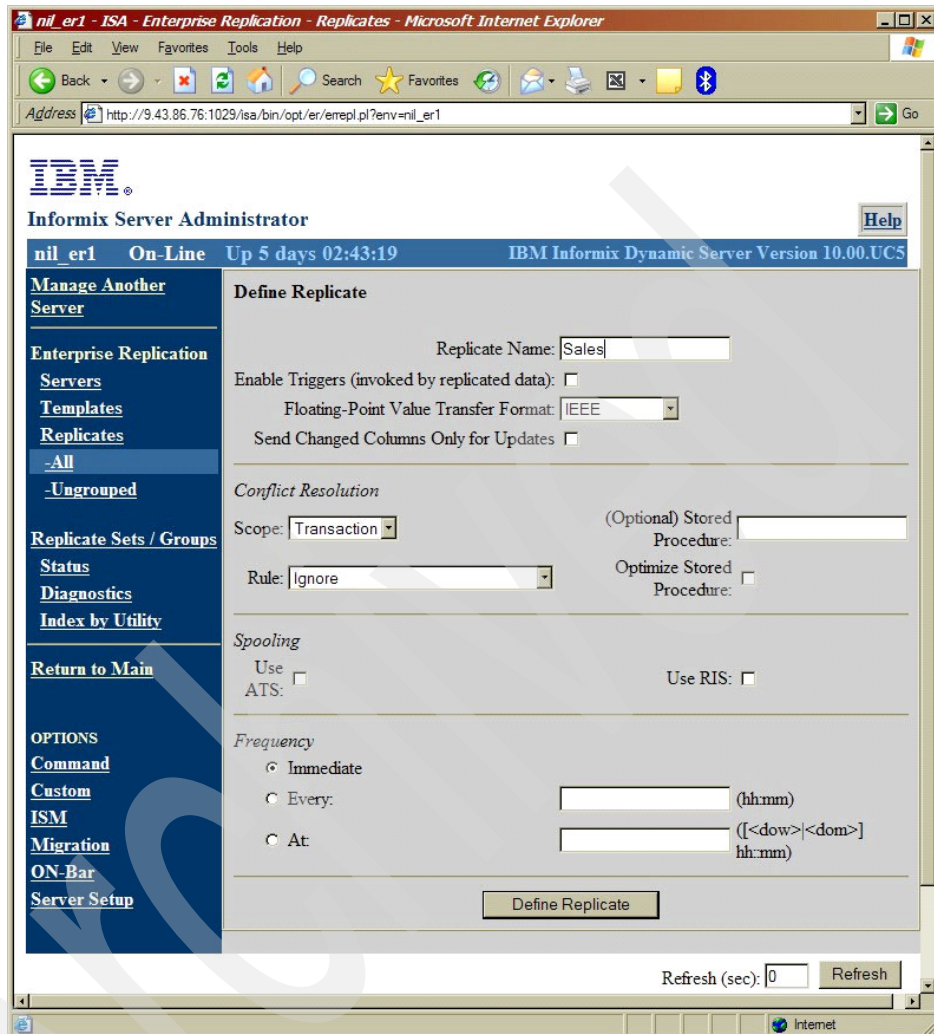


Figure 6-2 Defining a replicate using ISA (first window)

Figure 6-3 shows the window listing the replicates after it has been added. Note that there are no participants. Participants can be added by clicking the **Add Participant** button.

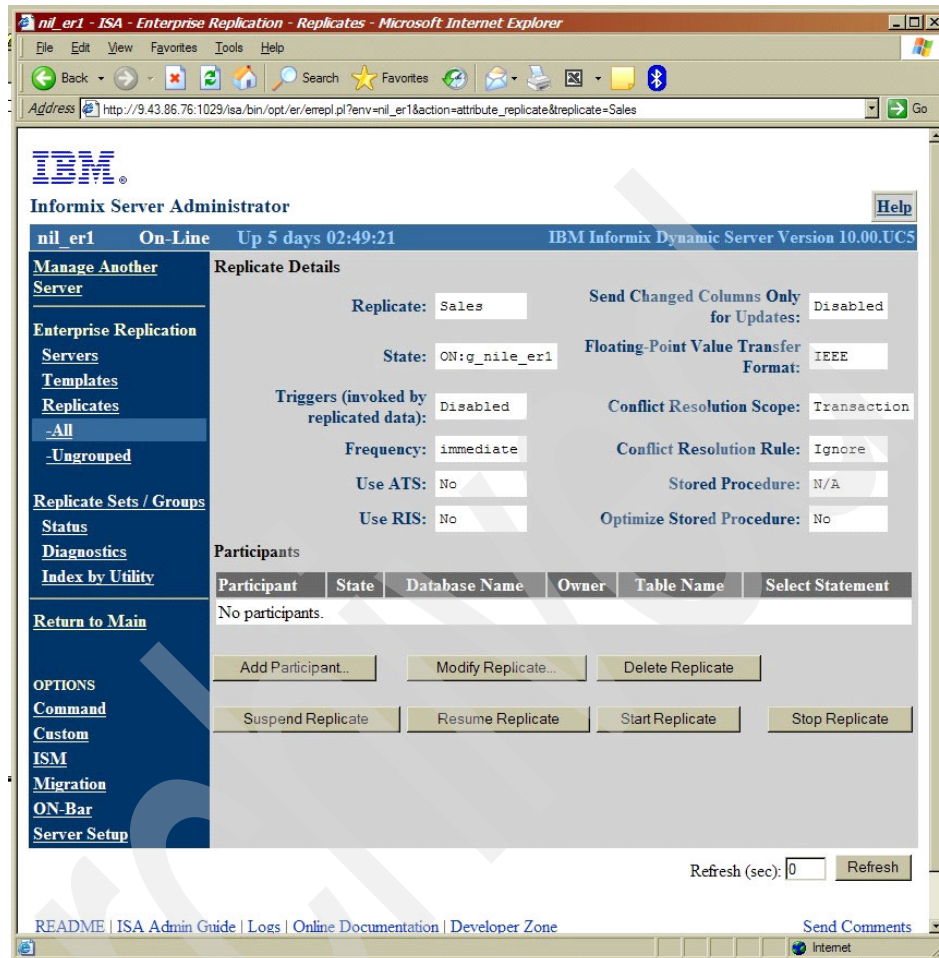


Figure 6-3 Listing a replicate

Figure 6-4 shows the window used to add participants to a replicate. There are windows for entering all the properties of the replicate, such as database, table, and select statement.

Thus, using ISA to define a replicate requires two steps, one to define the replicate properties, and a second to specify the database servers and tables that are used. Further, each replicate requires this sequence of multiple steps. That may be more time-consuming than writing and executing a script using the command-line interface. However, ISA may be a more useful way to become familiar with the ER administrative processes.

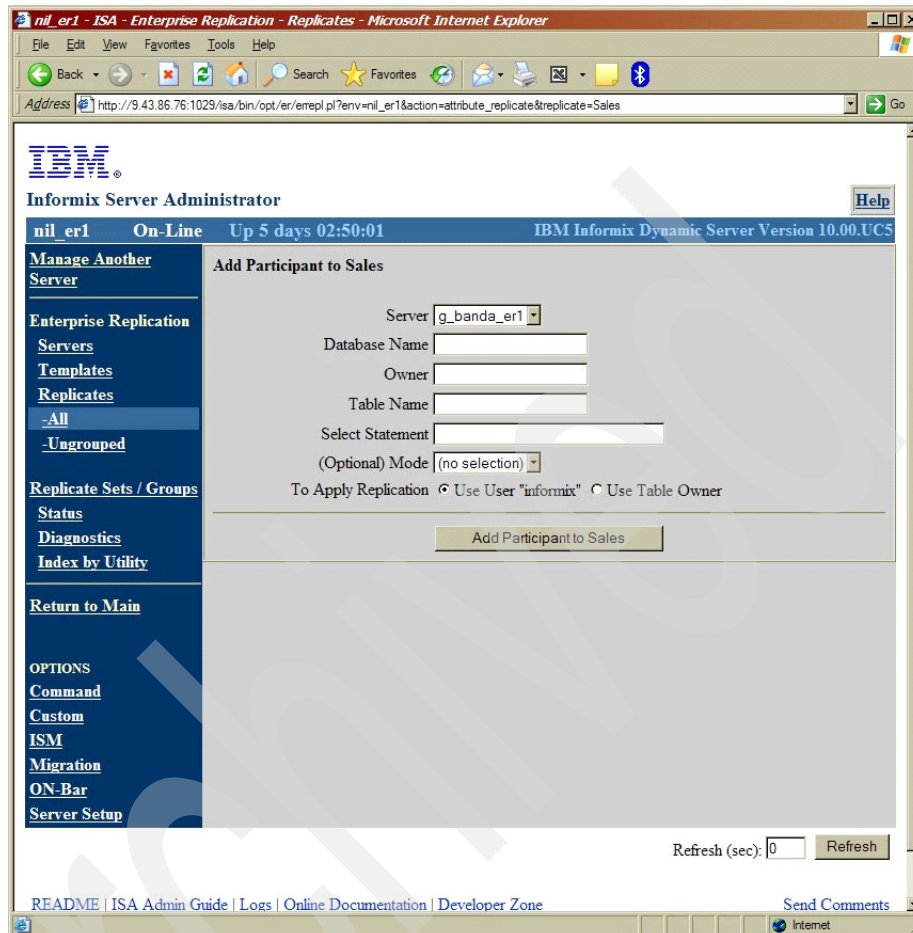


Figure 6-4 Adding participants with ISA

6.3.4 Defining replicate sets

In many cases, there will be groups of replicates that need to be managed as a unit. For instance, in the example database, replicating daily sales information means replicating both the items and orders tables. Doing one without the other does not provide consistent data, so it make sense to define a single replicate set containing those two replicates. That means the set can be stopped and started as a single unit, thus there less risk that an administrator would do one and fail to do the other. However, simply having the two replicates in the set does not prevent operations on the replicates individually. The set just makes it easier to work on multiple replicates at the same time.

A replicate set may be exclusive or non-exclusive. An exclusive replicate set means that the replicates are in only that set and no others. A replicate may be in more than one non-exclusive replicate set.

Example 6-4 shows the definition of a replicate set containing the customer, orders, and item replicates. This is an exclusive set. Each member replicate may not be in any other replicate set.

Example 6-4 Defining a Replicate Set

```
banda:/home/informix $ cat repset1.sh
cdr define replicateset --exclusive ordset \
nilew_cust bandaw_cust niler_cust bandar_cust \
nileitems bandaitems nileorders bandaorders

banda:/home/informix $ . ./repset1.sh

banda:/home/informix $ cdr list replset
Ex T REPLSET                                PARTICIPANTS
-----
Y  N ordset                                bandaitems, bandaorders,
                                           bandar_cust, bandaw_cust,
                                           nileitems, nileorders,
                                           niler_cust, nilew_cust
```

6.4 Starting enterprise replication

When a replication server is defined, it is started by default. When a replicate is defined, it is *not started* by default. In order for a replicate to actually transfer data, it must be started with the **cdr start replicate** command.

The process is simple. Each replicate is started with the **cdr start replicate** command. At that time, the data is synchronized among the servers, and after that, changes are propagated according to the replicate definitions.

In our example, the data from the orders table on each leaf node is sent to the root node but nowhere else. So after an order updates on the branch2 leaf, the change shows up on the HQ server but not on the g_nil_er2 server.

Replicates can 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 using the **cdr resume server** command, then the accumulated data is

sent. When a stopped server is restarted with the `cdr start` command, everything since the server was stopped is re-evaluated and sent as appropriate.

6.5 Templates

A template is a step beyond either a replicate or a replicate set. It is a way to define how data should be replicated without actually creating the replicates or any sets. Later on, when the replicate is *realized*, the replicates and sets are created automatically. The process to realize a replicate is discussed in 6.5.3, “Realizing templates” on page 85.

The value of templates is that a single definition can be created and then used to create the required replicates on the participating servers without the risk of making an error on one server that goes unnoticed. Everything will be set up identically or not at all. If there are a significant number of replication servers to manage, templates simplify the work significantly.

Just defining the template does not actually do anything on, with, or to, any IDS instance. A template is just a specification or definition. Changes are made to the IDS instances only when the template is realized.

For additional information about templates, see the article on IBM developerWorks® at the following URL:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0505kedia/index.html>

6.5.1 Terms and conditions

Templates cannot be used for all or for very 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 once 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`). This constraint means the requirements from 6.1, “The examples” on page 66, cannot be met using templates. However, if the requirements are relaxed to include the entire stock and customer tables, then templates can be used.

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.

6.5.2 Defining a template

Even with all those conditions, templates are a convenient way to set up many replication schemes.

Example 6-5 shows the **cdr define template**. In this example the template is created to match the requirements in 6.1, “The examples” on page 66, but using all the stock and customer tables. This is simpler than setting up and starting the corresponding replicate definition on each system.

A number of things about this example should be noted:

1. The templates options are applied to all the tables. If your requirements do not allow that then multiple templates are required. If some tables require options different from the rest, those tables cannot all be in one template.
2. A *mastered replicate* is defined. That means the replicate on each system where the template is realized *will* use the definition on HQ where the template is defined.
3. 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.
4. The example shows the **list template** command to confirm the existence of the template. This is not required, it is just a convenient way to be sure things were created as expected.

Example 6-5 Defining replication templates

```
banda:/home/informix $ cat deftempl.sh
cdr define template sales -C ignore -S trans --master=HQ
--database=stores \
customer orders items manufact
banda:/home/informix $ . ./deftempl.sh
Obtaining dictionary for stores@HQ:'informix'.customer
Obtaining dictionary for stores@HQ:'informix'.orders
Obtaining dictionary for stores@HQ:'informix'.items
Obtaining dictionary for stores@HQ:'informix'.manufact
Creating mastered replicate sales_ban_er1_4_1_customer for table
'informix'.customer
Creating mastered replicate sales_ban_er1_4_2_orders for table
'informix'.orders
Creating mastered replicate sales_ban_er1_4_3_items for table
'informix'.items
```

```
Creating mastered replicate sales_ban_er1_4_4_manufact for table
''informix''.manufact
```

```
banda:/home/informix $ cdr list template sales
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
REPLICATE:   sales_ban_er1_4_1_customer
OWNER:       informix
TABLE:       customer:x:x
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
REPLICATE:   sales_ban_er1_4_2_orders
OWNER:       informix
TABLE:       orders
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
REPLICATE:   sales_ban_er1_4_3_items
OWNER:       informix
TABLE:       items
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
REPLICATE:   sales_ban_er1_4_4_manufact
OWNER:       informix
TABLE:       manufact
```

```
banda:/home/informix $
```

6.5.3 Realizing templates

When a set of replication servers is defined and started, templates can be realized to install the replicate definitions. This is usually simpler than defining

and starting each corresponding replicate server, particularly when there is a large number of servers.

The command to realize a template (**cdr realize template**) does a number of things, such as:

1. Adding one or more participants to the replicate definition
2. Creating tables if necessary
3. Populating the tables or synchronizing the data with the master data
4. Starting the replicate set on the realized servers

Note: If a database is going to be created and populated, be sure there is enough room for it. If not, the command will fail.

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 user informix).

Example 6-6 shows the realize command in action. The intent is to realize the template on **branch1**, the leaf server connected to the root. That requires a number of operations:

1. Before any of this can be done, the database must be created in the target instance. ER will create tables but not databases. That is not shown in the example, but it can be done either with **dbaccess** or ISA.
2. 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 will fail because the source for the data will not be known.
3. 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.

Notice that difference. The first lines of output (those referring to HQ) are from the first command (to realize the template on HQ). The tables exist, so once their properties are confirmed, no other work is necessary. The output of the second command (those referring to branch1) demonstrate that the tables were not found and are being created.

Realizing a template creates replicates for all the tables in the template on each system specified in the command to realize the template. That means that as systems are added to the replication network, the work to set up the replicates is simplified to realizing a set of templates.

Example 6-6 Using templates to create replicas

```
banda:/home/informix $ cat realizetempl.sh
```

```
cdr realize template sales HQ
```

```
cdr realize template sales -c HQ --autocreate --syncdatasource=HQ
--extratargetrows=keep --target branch1
```

```
banda:/home/informix $ . ./realizetempl.sh
```

```
Verification of stores@HQ:'informix'.customer started
```

```
Verification of stores@HQ:'informix'.customer is successful
```

```
Verification of stores@HQ:'informix'.orders started
```

```
Verification of stores@HQ:'informix'.orders is successful
```

```
Verification of stores@HQ:'informix'.items started
```

```
Verification of stores@HQ:'informix'.items is successful
```

```
Verification of stores@HQ:'informix'.manufact started
```

```
Verification of stores@HQ:'informix'.manufact is successful
```

```
Verification of stores@branch1:'informix'.customer started
```

```
Creating table...
```

```
create table 'informix'.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@branch1:'informix'.customer is successful
```

```
Verification of stores@branch1:'informix'.orders started
```

```
Creating table...
```

```
create table 'informix'.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@branch1:'informix'.orders is successful

Verification of stores@branch1:'informix'.items started

Creating table...

```

create table 'informix'.items (
    item_num smallint,
    order_num integer,
    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@branch1:'informix'.items is successful

Verification of stores@branch1:'informix'.manufact started

Creating table...

```

create table 'informix'.manufact (
    manu_code char(3),
    manu_name char(15),
    lead_time interval day(3) to day,
    primary key (manu_code));

```

Verification of stores@branch1:'informix'.manufact is successful

banda:/home/informix

\$

This command has a number of other options, and all of them are described in the Guide to Enterprise Replication.

Note: The real power of creating and realizing templates is the ability to define everything in one place and apply the definitions in multiple places. For many configurations and requirements, this is far simpler than the work of doing things separately on each server.

6.6 Other replication considerations

In this section we discuss and describe other replication considerations.

6.6.1 Blocking replication in a transaction

At times it may be necessary to perform an operation without sending the changes to other systems. The `WITHOUT REPLICATION` clause in the `BEGIN WORK` statement starts a transaction for which no changes will be replicated. Be careful using this. Improper changes may make the databases inconsistent and require a repair operation.

6.6.2 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. Since 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 are to 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, see the article on IBM developerWorks at the following URL:

<http://www.ibm.com/developerworks/db2/zones/informix/library/techarticle/0208pincheira/0208pincheira.html>

Making changes in enterprise replication

In this chapter we demonstrate several scenarios where replication servers, replicates, and replicate sets require modifications due to changing business requirements.

The definitive guide for all enterprise replication commands is the *IBM Informix Dynamic Server Enterprise Replication Guide*, G251-2279.

7.1 Scenarios for making changes to existing ER

Here we present a few scenarios where an already existing replication server, replicate, and replicate set require some modifications. These are scenarios that represent typical business needs when using enterprise replication.

To test the scenarios, we implemented a replication environment that was used for the various tests and demonstrations described in this book. The environment included 3 servers and the Informix Stores Example Database, described in detail in Chapter 2, “Preparing the technical infrastructure” on page 7. The topology for the environment is depicted in Figure 7-1.

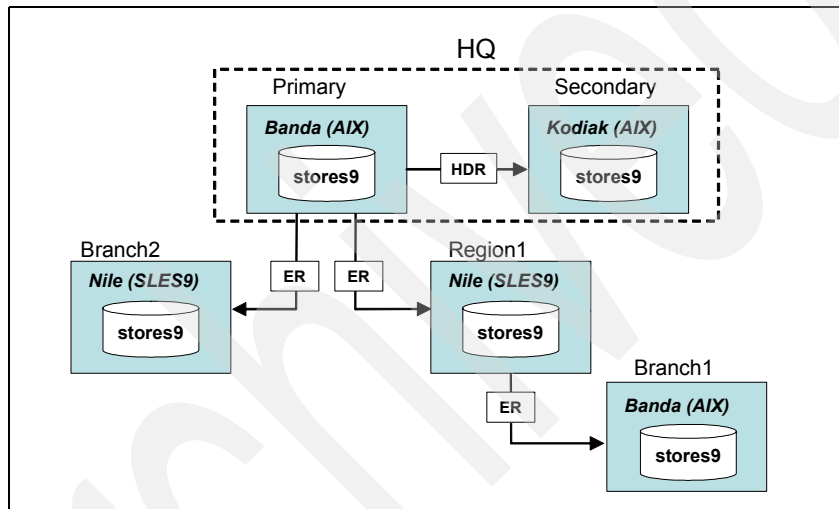


Figure 7-1 Project environment topology

7.2 Changing a replication server

Due to changing business requirements, replication server topologies may require the addition, modification, and deletion of replication servers. Since replication servers are the replication gatekeepers, these objects also need to be started, stopped, suspended, and resumed.

Finding replication servers

Banda is the root node of the environment, so we can enter commands there to find the information desired. For example, to list the replication servers that have been defined thus far, we used the following list server command:

```
% cdr list server
```

This results in a listing of all the replication servers that have been defined, as shown in Example 7-1.

Example 7-1 Environment server listing

```
test
banda:/home/informix $ cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED

HQ	100	Active	Local	0		
Branch1	101	Active		0		
Branch2	300	Active	Connected	0	Sep 28 11:39:47	
Region1	301	Active	Connected	0	Sep 28 11:39:40	

View an individual server

You can add a server group name to the **cdr list server** command to view an individual server, as illustrated in Example 7-2.

Example 7-2 cdr list server <server group name>

```
banda:/home/informix $ cdr list server region1
```

NAME	ID	ATTRIBUTES

Region1	301	timeout=500 root=g_hq leaf

View a server via a global catalog on another server

The **cdr list server** command also takes the option: **--connect=<ids server>** to list servers from a global catalog on another server. This is illustrated in Example 7-3.

Example 7-3 cdr list server --connect=<ids server>

```
banda:/home/informix $ cdr list server --connect=ban_er2
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED

Branch1	101	Active	Local	0		
Branch2	300	Active	Connected	0	Sep 28 11:39:58	

Modifying a replication server

When we created the replication servers, some of them timed out due to a low idle time. To modify any of the servers to increase the idle timeout, use the procedure as illustrated in Example 7-4. Note that the first update to the Branch1 node received an error. Why? The Branch1 node is a leaf node and does not

have a complete catalog. We then updated the Region1 node to which the Branch1 node is connected with the following command:

```
% cdr modify server
```

This 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. Example 7-4 shows how we changed the idle timeout and shows the modified idle timeout.

Example 7-4 *cdr modify server <server group name>*

```
banda:/home/informix $ cdr modify server -c ban_er2 -i 700 branch1
command failed -- request denied on limited server (75)
banda:/home/informix $ cdr modify server -c nil_er1 -i 700 branch2
banda:/home/informix $ cdr list server branch2
```

NAME	ID	ATTRIBUTES

Branch2	300	timeout=700 root=g_hq hub

Stopping and restarting a replication server

If the goal is to completely stop ER, the best way to accomplish it is to shut down the server. However, if you need to temporarily stop ER without bringing the server down, use the **cdr stop server** command. Stopping the ER server will stop capturing data to be replicated.

% *cdr stop*

As demonstrated in Example 7-5, we stopped cdr from the HQ node and checked to see if the node has been stopped from the Region1 node.g_hq.

Example 7-5 *cdr stop*

```
banda:/usr/informix/etc $ cdr stop
banda:/usr/informix/etc $ cdr list server
```

Checking whether ER on HQ node has been stopped from Region1 node:

```
informix@nile:~> cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED

g_hq	100	Active	Dropped	0	Sep 28	15:12:28
g_branch1	101	Active	Connected	0	Sep 28	11:49:16
g_region1	300	Active	Local	0		
g_branch2	301	Active		0		

% **cdr start**

To restart a stopped ER server, use the **cdr start** command. This is shown in Example 7-6. As you can see in the example, the command failed. However, this is because we issued it without first stopping the server, so it was already started.

Example 7-6 *cdr start*

```
banda:/usr/informix/etc $ cdr start
banda:/usr/informix/etc $ cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED
HQ	100	Active	Local	0		
Branch1	101	Active		0		
Branch2	300	Active	Connected	0	Sep 28 15:06:10	
Region1	301	Active	Connected	0	Sep 28 15:06:10	

```
command failed -- Enterprise Replication already active (63)
```

Suspending and resuming a replication server

Suspending an ER server differs from stopping it in that the source server queues replicate data, but suspends replicating the data to the target servers. Other background information such as network acknowledgement and control messages will continue and not be affected by the suspension.

% **cdr suspend server <server group name>**

Notice in Example 7-7 that when we suspend g_branch2 from g_hq the cdr list server output does not show any differences. However, checking the Branch2 node directly shows the suspension. When suspending a server, make sure that your send queues 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 *Informix Enterprise Replication Guide*. There is also more information about this topic in Chapter 5, “Defining the enterprise replication topology” on page 55.

Example 7-7 *cdr suspend server*

```
banda:/usr/informix/etc $ cdr suspend server HQ region1
banda:/usr/informix/etc $ cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED
HQ	100	Active	Local	0		
Branch1	101	Active		0		
Branch2	300	Active	Connected	0	Sep 28 15:06:10	
Region1	301	Active	Connected	0	Sep 28 15:06:10	

```
informix@nile:~> cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED
HQ	100	Suspend	Connected	0	Sep 28	15:15:28
Region1	301	Active	Local	0		

Deleting a replication server

Even after careful topology planning, there will be times when a replication server is no longer needed. The command that enables you to delete a server is:

```
% cdr delete server
```

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, it is highly recommended that you review your topology to ensure that the delete will not affect any other nodes. Also, it is recommended that you do not delete a replication server and then immediately recreate a replication server with the same name.

As an example, when we created our topology, banda_er1 connected to Branch2, which also had a leaf node attached to it called Branch1. When we tried to delete branch2, ER let us know that a child for the server Branch2 exists on other ER servers and that we cannot drop it. We dropped the child and then the parent server, as depicted in Example 7-8.

Example 7-8 *cdr delete server*

```
banda:/home/informix $ cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED
HQ	100	Active	Local	0		
Branch1	101	Active		0		
Branch2	300	Active	Connected	0	Nov 13	12:21:16
Region1	301	Active	Connected	0	Nov 13	12:24:08

```
banda:/home/informix $ cdr delete server region1
command failed -- cannot delete server with children (72)
```

```
banda:/home/informix $ cdr delete server branch1
banda:/home/informix $ cdr delete server branch1
banda:/home/informix $ cdr list server
```

SERVER	ID	STATE	STATUS	QUEUE	CONNECTION	CHANGED
HQ	100	Active	Local	0		
Branch2	300	Active	Connected	0	Nov 13	12:21:16

```

Region1          301  Active  Connected    0    Nov 13 12:24:08

banda:/home/informix $ cdr delete server --connect=ban_er2 branch2
banda:/home/informix $ cdr delete server branch2
banda:/home/informix $ cdr list server
SERVER            ID    STATE    STATUS      QUEUE    CONNECTION CHANGED
-----
HQ                100  Active  Local       0
Region1          301  Active  Connected   0    Nov 13 12:24:08

```

7.3 Changing replicates

Many times just the replicates need fine tuning, or a business request comes in that requires that only a replicate needs to be modified. In this section, we demonstrate several ways to find and modify replicates.

As a refresher, a replicate contains attributes or participants, properties, and state, and all these objects can be modified. In our case, we defined one of the replicates on the orders table to be every row (select * from orders). Then we decided to modify the replicate to be only a few columns rather than the entire table. The first thing we needed to do was to find the name of the replicate.

List replicates

To find all the replicates we defined on the instance ban_er1, we issued the following command:

```
% cdr list replicate
```

By executing this command and not specifying the name of a replicate, all replicates defined on the server are listed. In Example 7-9, we show a partial list of the many replicates that exist on the server.

Example 7-9 cdr list replicate

```

banda:/home/informix $ cdr list replicate

CURRENTLY DEFINED REPLICATES
-----
REPLICATE:      prices
STATE:          Active ON:HQ
CONFLICT:        Ignore
FREQUENCY:       immediate
QUEUE SIZE:     0
PARTICIPANT:     stores9:informix.stock

```

```

OPTIONS:          transaction,fullrow
REPLID:           6553601 / 0x640001
REPLMODE:         PRIMARY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

REPLICATE:        bandaorders
STATE:            Inactive ON:HQ
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.orders
OPTIONS:          transaction,fullrow
REPLID:           6553602 / 0x640002
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

REPLICATE:        bandaitems
STATE:            Inactive ON:HQ
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.items
OPTIONS:          transaction,fullrow
REPLID:           6553604 / 0x640004
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ
...

```

As can be seen in the list, there is a replicate by the name of *bandaorders*. To obtain more information about this replicate we used the list replicate brief command:

```
% cdr list replicate brief
```

This command allows you to see a summary of all the replicates that have been defined, and is depicted in Example 7-10. As you look at the list, you will notice that there are duplicate replicate names. Each replicate name lists the participants in the replicate. The **cdr list replicate brief** command shows a partial list of the replicates. Notice that the *bandaorders* replicate has two participants.

Example 7-10 cdr list replicate brief

banda:/home/informix \$ **cdr list replicate brief**

REPLICATE	TABLE	SELECT STATEMENT
prices	stores9@HQ:informix.stock	select stock_num,
	manu_code, unit, unit_price from stock	
prices	stores9@branch1:informix.stock	select stock_num,
	manu_code, unit, unit_price from stock	
prices	stores9@branch2:informix.stock	select stock_num,
	manu_code, unit, unit_price from stock	
prices	stores9@region1:informix.stock	select stock_num,
	manu_code, unit, unit_price from stock	
bandaorders	stores9@HQ:informix.orders	select * from orders
bandaorders	stores9@branch1:informix.orders	select * from orders
nileorders	stores9@HQ:informix.orders	select * from orders

banda:/home/informix \$ **cdr list replicate brief bandaorders**

REPLICATE	TABLE	SELECT
bandaorders	stores9@HQ:informix.orders	select * from orders
bandaorders	stores9@branch1:informix.orders	select * from orders

Modifying replicates

In our scenario, we found the replicate that we wanted to change (bandaorders), which had a **select * from orders** statement. We wanted to change it to select only a few columns.

There are two ways to make changes to a replicate. To make the change we desired, the command **cdr change replicate** was used. This should not be confused with the other command, **cdr modify replicate**, which makes changes only to the replicate attributes such as conflict options, scope options, frequency options, and mode.

% cdr change replicate

This command enables you to change the select statement, but the only way to do so is to drop and add the participant using the correct select statement. Example 7-11 demonstrates how we altered the replicate.

Example 7-11 cdr change replicate

```
banda:/home/informix $ cdr list replicate brief bandaorders
```

REPLICATE	TABLE	SELECT
bandaorders	stores9@HQ:informix.orders	select * from orders

```
banda:/home/informix $ cdr change repl -d bandaorders  
"stores9@HQ:informix.orders"
```

```
banda:/home/informix $ cdr list replicate bandaorders
```

DEFINED REPLICATES ATTRIBUTES

```
-----  
REPLICATE:      bandaorders  
STATE:          ON:HQ  
CONFLICT:       Ignore  
FREQUENCY:      immediate  
QUEUE SIZE:     0  
PARTICIPANT:  :.  
OPTIONS:        transaction,fullrow  
REPLID:         6553621 / 0x640015  
REPLMODE:       OTHER () ON:HQ  
APPLY-AS:       OWNER ON:HQ
```

```
banda:/home/informix $ cdr list replicate brief bandaorders
```

REPLICATE	TABLE	SELECT
bandaorders	stores9@HQ:informix.orders	select * from orders

```
banda:/home/informix $ cdr change replicate -a mikeorders  
"stores9@HQ:informix.orders" "select  
order_num, order_date, customer_num from orders"
```

```
banda:/home/informix $ cdr list replicate brief bandaorders
```

REPLICATE	TABLE	SELECT
bandaorders	stores9@HQ:informix.orders	select order_num, order_date, customer_num from orders

7.4 Changing replicate sets

Replicate sets allow you to group many replicates together for easier management. By grouping many 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.

Listing replicate sets

To determine which replicate sets exist and which replicates are included within each set, use the `cdr list replicateset` command.

% *cdr list replicateset*

This command shows a summary of all the replicate sets created. This is depicted in Example 7-12, which shows the output of the replicate sets called `ordset` and `sales`. The first column of the output, **Ex** indicates whether the replicate set is exclusive. **Y** indicates yes, the replicate set is exclusive; **N** indicates that the replicate set is non-exclusive. If a replicate set is exclusive, the individual replicates can only be managed as part of the set.

The second column, **T** indicates whether the replicate set was created using a template.

To see a more detailed view of a particular replicate set, add the replicate set name at the end of the command as show in Example 7-12.

Example 7-12 *cdr list replicateset*

```
banda:/home/informix $ cdr list replicateset
```

Ex	T	REPLSET	PARTICIPANTS
Y	N	ordset	bandaitems, bandaorders, bandaw_cust, nileitems, nileorders, nilew_cust
N	Y	sales	sales_ban_er1_4_1_customer, sales_ban_er1_4_2_orders, sales_ban_er1_4_3_items, sales_ban_er1_4_4_manufact

```
banda:/home/informix $ cdr list replicates ordset
```

```
REPLICATE SET:ordset
```

CURRENTLY DEFINED REPLICATES

```
-----
REPLICATE:      bandaitems
STATE:          Active ON:HQ
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    stores9:informix.items
OPTIONS:        transaction,fullrow
REPLID:         6553604 / 0x640004
REPLMODE:       READ-ONLY ON:HQ
APPLY-AS:       INFORMIX ON:HQ

REPLICATE:      bandaorders
STATE:          Active ON:HQ
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    stores9:informix.orders
OPTIONS:        transaction,fullrow
REPLID:         6553602 / 0x640002
REPLMODE:       READ-ONLY ON:HQ
APPLY-AS:       INFORMIX ON:HQ

REPLICATE:      bandaw_cust
STATE:          Active ON:HQ
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    stores9:informix.whlsale_customer
OPTIONS:        transaction,fullrow
REPLID:         6553608 / 0x640008
REPLMODE:       READ-ONLY ON:HQ
APPLY-AS:       INFORMIX ON:HQ
```

Modifying replicate sets

Replicate sets can only be modified by adding, deleting replicates within the replicate set, or changing the replication frequency. Remember that when modifying any replicate set, every replicate in the set will be affected.

% *cdr change replicateset*

We defined a replicate set called *sales* and now want to add a new replicate. When the replicate set was created, we had the choice of creating it as a exclusive set or non-exclusive set. Exclusive replicate sets are used when

replicates within the set include tables that have referential constraints. The default is non-exclusive replicate set.

A new branch has been proposed and as a test, the company wants to begin sharing information with the new branch. We changed the ordset to add the new branch in Example 7-13.

Example 7-13 cdr change replicateset

```
banda:/home/informix $ cdr change replicateset --add ordset nilew_cust  
nileitems nileorders
```

```
banda:/home/informix $ cdr list replicates ordset
```

```
REPLICATE SET:ordset
```

CURRENTLY DEFINED REPLICATES

```
-----  
REPLICATE:      nileitems  
STATE:          Active ON:HQ  
CONFLICT:       Ignore  
FREQUENCY:      immediate  
QUEUE SIZE:     0  
PARTICIPANT:    stores9:informix.items  
OPTIONS:        transaction,fullrow  
REPLID:         6553605 / 0x640005  
REPLMODE:       READ-ONLY ON:HQ  
APPLY-AS:       INFORMIX ON:HQ
```

```
REPLICATE:      bandaitems  
STATE:          Active ON:HQ  
CONFLICT:       Ignore  
FREQUENCY:      immediate  
QUEUE SIZE:     0  
PARTICIPANT:    stores9:informix.items  
OPTIONS:        transaction,fullrow  
REPLID:         6553604 / 0x640004  
REPLMODE:       READ-ONLY ON:HQ  
APPLY-AS:       INFORMIX ON:HQ
```

```
REPLICATE:      bandaorders  
STATE:          Active ON:HQ  
CONFLICT:       Ignore  
FREQUENCY:      immediate  
QUEUE SIZE:     0  
PARTICIPANT:    stores9:informix.orders
```

```

OPTIONS:          transaction,fullrow
REPLID:           6553602 / 0x640002
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

REPLICATE:        nileorders
STATE:            Active ON:HQ
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.orders
OPTIONS:          transaction,fullrow
REPLID:           6553603 / 0x640003
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

REPLICATE:        bandaw_cust
STATE:            Active ON:HQ
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.whlsale_customer
OPTIONS:          transaction,fullrow
REPLID:           6553608 / 0x640008
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

REPLICATE:        nilew_cust
STATE:            Active ON:HQ
CONFLICT:         Ignore
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      stores9:informix.whlsale_customer
OPTIONS:          transaction,fullrow
REPLID:           6553609 / 0x640009
REPLMODE:         READ-ONLY ON:HQ
APPLY-AS:         INFORMIX ON:HQ

```

Starting and stopping versus suspending a replicate set

After a replicate set has been started, it is not recommended that you stop or suspend it, which basically stops and suspends replicates. Even if the replicate set is created as an exclusive set, stopping or suspending a replicate set is not recommended. In the 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. Once a stop or suspend command is issued, ER will stop sending information even if it is within a transaction. For instance, 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 once replication has resumed all the pieces will be sent in the correct order. Therefore, children rows could arrive before the parent row. If Referential Integrity is turned on, you will receive an error and the child will be rolled back. In addition, the parent part of the transaction could come later and it could be successfully replicated, creating incorrect data. Referential Integrity constraints are the primary reason why it is not recommended to stop or suspend a replicate or replicate set.

What is recommended, and is considered a best practice using enterprise replication, is to let ER take care of itself with careful planning and monitoring. In the planning phase, make sure that you have enough queue space set aside for the specific situation where a replication server is down.

How much data is replicated per hour and how many hours are going to be allowed to queue will dictate how much queue space you will need. Once replication is resumed, re-syncing the down server is quite fast. We have seen cases with a financial company that had a site down for 3 to 5 hours, with thousands of transactions queuing up. Once 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, it is recommended to delete the replication server altogether using the **cdr delete server** command, and then re-sync using the default **online sync** or **cdr check sync** command.

Another solution to address a down replication server is to create replicates and replicate sets using templates. When you delete a server there can be many replicates that need to be re-created. If you do not use templates, this can be a very tedious task. Using templates, all the replicates and replicate sets are created and just need to be realized using the **cdr realize template** command, and all the replicates that have been deleted will be created again.

If you still plan to stop or suspend a replicate or replicate set, refer to the *IBM Informix Enterprise Replication Guide*, G251-2279 for details.

7.5 Template changes

Templates make administering enterprise replication much easier. Templates provide a simple way to set up and deploy a large set of tables for enterprise replication. Once a template is created, it will set up replicates, perform initial data synchronization, and even create 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
```

Listing a template

Once a template is created, you can see what it consists of using the `% cdr list template` command.

% cdr list template

This command will list all templates created. If more detail about a particular template is needed, simply add the name of the template at the end of the command, as shown in Example 7-14.

Example 7-14 cdr list template

```
banda:/home/informix $ cdr list template
```

TEMPLATE	DATABASE	TABLES
=====	=====	=====
sales	stores	informix.customer
	stores	informix.items
	stores	informix.manufact
	stores	informix.orders

```
banda:/home/informix $ cdr list template sales
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
REPLICATE:   sales_ban_er1_4_1_customer
OWNER:       informix
TABLE:       customer
```

```
TEMPLATE:    sales
TEMPLATE ID: 6553611
SERVER:      HQ
DATABASE:    stores
```



```

REPLICATE:  sales_ban_er1_4_2_orders
OWNER:      informix
TABLE:      orders

TEMPLATE:   sales
TEMPLATE ID: 6553611
SERVER:     HQ
DATABASE:   stores
REPLICATE:  sales_ban_er1_4_3_items
OWNER:      informix
TABLE:      items

TEMPLATE:   sales
TEMPLATE ID: 6553611
SERVER:     HQ
DATABASE:   stores
REPLICATE:  sales_ban_er1_4_4_manufact
OWNER:      informix
TABLE:      manufact

```

```
banda:/home/informix $
```

Deleting a template

If a template is in need of any changes, the only way to make them is to first delete the template and then re-create it. This is why it is important to save all the cdr commands in a script file. When needed, you can simply re-execute the script file.

% *cdr delete template* <templatename>

In Example 7-15, we demonstrate how easily a template can be deleted with this command.

Example 7-15 cdr delete template

```
banda:/home/informix $ cdr list template
```

TEMPLATE	DATABASE	TABLES
=====	=====	=====
sales	stores	informix.customer
	stores	informix.items
	stores	informix.manufact
	stores	informix.orders

```
banda:/home/informix $ cdr delete template sales
```

```
banda:/home/informix $ cdr list template
```

TEMPLATE	DATABASE	TABLES
=====		
banda:/home/informix \$		

Monitoring and troubleshooting ER

Enterprise replication in IDS is one of the most powerful replication technologies available in any database server. Enterprise replication also provides a highly flexible replicate definition. The routine and topology enables many choices for implementing any type of data replication requirement, coupled with a complex and secure routing network. Because it works with heterogeneous hardware and a number of operating systems, it is capable of being widely used in most circumstances. Apart from all these properties, ER also works between different versions of IDS. As an example, a replication environment can be set up between IDS version 9.4 and IDS version 10.00.

Any software that provides such flexibility, coupled with high performance, has to be highly sophisticated. In such sophisticated and complex environments, problems are to be expected. This is not to say that the troubles that arise are due to ER; problems can come from the operating system, file system, network malfunction, and so forth. These are all problems that can hamper the operation of ER.

In spite of its sophistication and complexity, ER is widely used because of its robustness and ease of troubleshooting. One of the very well known commands used by Informix DBAs is **onstat**. The ER support for that command has been extended and provides a number of options for debugging any type of replication network.

In this chapter we discuss some of the common problems that are faced by users of ER. These problems are often due to an incorrect configuration of the ER system or to external factors that are beyond the scope of ER to correct. ER stores all configuration information in a separate database, which resides inside each of the ER nodes. The database is known as the *syscdr* database.

One of the most common methods for diagnosing a problem in ER is by collecting various statistics. These statistics are provided by the *onstat* command, and are discussed throughout this chapter.

8.1 How enterprise replication works

To troubleshoot any replication issues, it is necessary to understand the working principle of the software. Enterprise replication is handled by a component of the IBM Informix database engine that provides the capability to replicate transactions from one database to another. For example, it is used for replicating data in user-created tables. ER does this by allowing the user to specify what data needs to be replicated. It maintains the list of tables and the replication strategies in its own database. This database is known as the Global catalog and is stored in database *syscdr*. The ER subsystem then looks through the log files to find those transactions that updated, inserted, or deleted data tagged as requiring replication. The Informix database threads that look through the log file are referred to as *snoopers*.

In Figure 8-1 we show all the threads and their relationships with each other. The snooper passes these log records to a set of threads which are known as *groupers*. These groupers filter the log records and prepare all committed transactions in an architecture-independent format for sending to the target database. These transactions are then handed over to the NIF (Network Interface) module to be sent to the target system. On the target server, these transactions are received by the NIF module, which then places them in another queue.

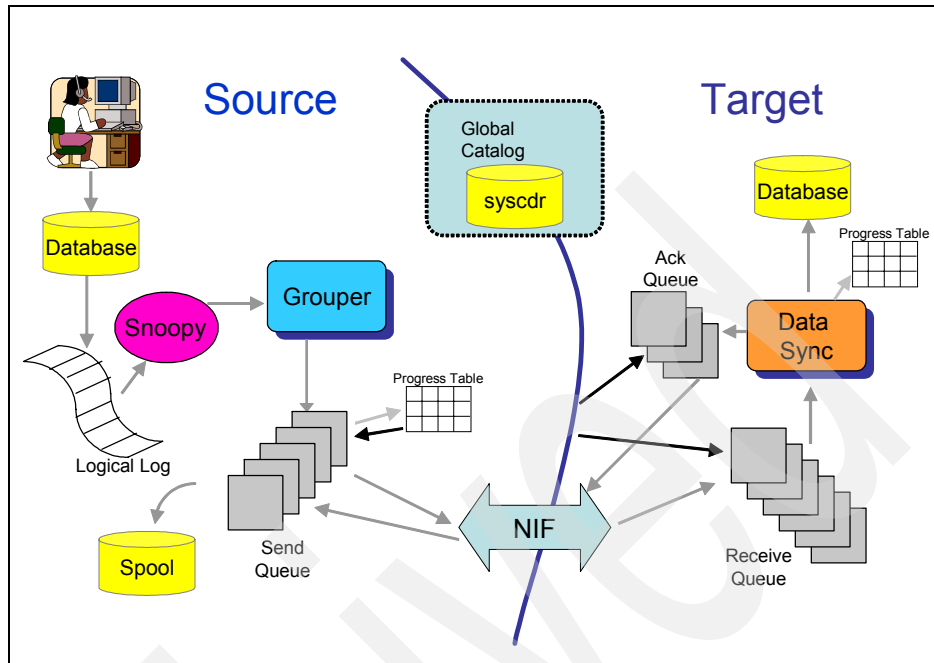


Figure 8-1 How ER works

The data sync subsystem on the target server picks up received transactions from queue and applies them to the target table. Each transaction received from the source server is applied as a single transaction. Almost every subsystem in an enterprise replication environment supports parallelism. So, transaction ordering is maintained at the destination sites, and parallelism is supported subject to data consistency restrictions. On both the source and target system, enterprise replication creates many progress tables. Progress tables provide information about the progress of each replicate under the headings: Server, Group, Bytes Queued, Acked, and Sent. Their meanings are as follows:

- ▶ **Group field:** shows the replicate ID
- ▶ **Acked field:** shows what has been acknowledged
- ▶ **Sent field:** shows which entries are now in transit

Both the Aacked and the Sent fields show the RQM key in a fixed format.

Based on the functionality, enterprise replication is broadly divided into the following logical components:

- ▶ **Global Catalog:** Manages metadata related to replication.
- ▶ **Log snooper:** Responsible for reading logical log files.

- ▶ **Grouper:** Regroups and queues transactions into a send queue for replication based on the metadata available in the Global Catalog (*syscdr* database).
- ▶ **Queuer:** Responsible for staging replicated transactions until they get 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.
- ▶ **Receive Manager:** Responsible for queuing replicated transactions to a 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.

8.2 Handling asynchronous communication of ER

All operation in ER are asynchronous in nature. This includes the control messages as well as 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 modify the *syscdr* database.

Asynchronous control messages can cause confusion to the users. 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, the user may think that the operation is successful, and that the next command, which may depend on the previous command, can be performed immediately. But this may result in error, which may be surprising to the users. Consider the following example. The four commands shown in Example 8-1 define 4 servers in a topology that is run from a script.

Example 8-1 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 8-1 may produce the output shown in Example 8-2 (when the last command is executed).

Example 8-2 ER command failed due to asynchronous messaging

```
$ cdr define server -c ws306sl -i 500 -L -S g_srv2 -I g_srv3  
command failed -- invalid server (47)
```

Another example of an error is one that happens when you drop and recreate an ER node. This is depicted in Example 8-3.

Example 8-3 Failure due to dropping and recreating 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 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.

8.3 Using onstat for ER statistics collection

One of the ways to monitor replication in IDS is by collecting the various statistics using the **onstat** command. Figure 8-2 illustrates some of the statistics collection options of onstat associated with various sub-components of enterprise replication.

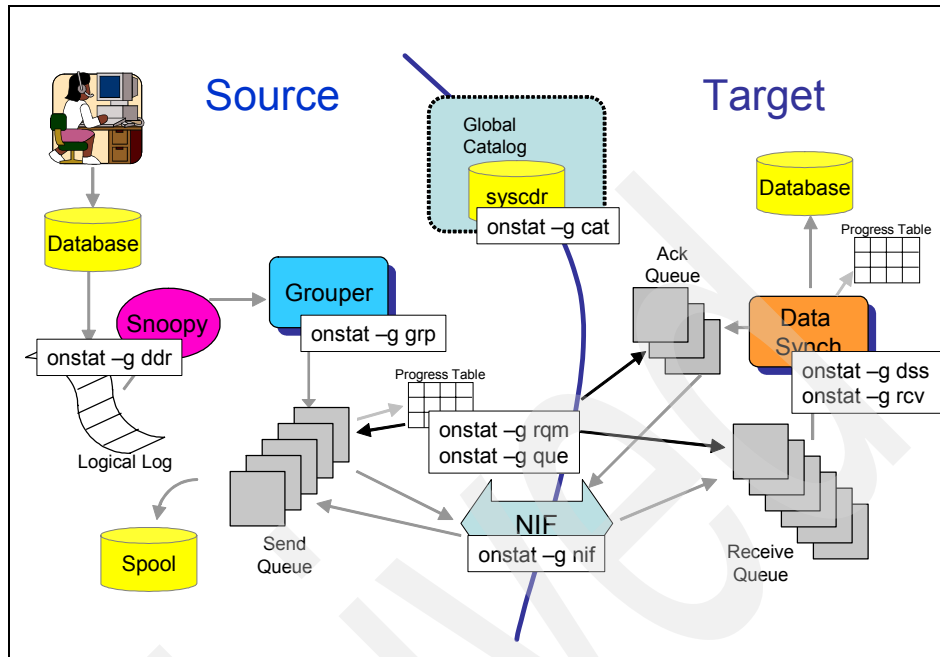


Figure 8-2 ER and onstat

Most common ER options of onstat produce very verbose output, and are self explanatory. Some of the outputs produced by these onstat commands are binary numbers that are useful for tech support professionals who are helping customers to diagnose the problem. They refer to internal data structures, which will be used later in analyzing the problem in depth. These can be ignored by DBAs and end users of ER.

8.3.1 Global Catalog statistics (onstat -g cat)

There are two types of global catalog dumps that are useful for users. The first one provides information about all servers in the ER network and the second lists replicate-related statistics.

Example 8-4 shows a sample of the servers statics. For this example there are four servers: g_srv1, g_srv2, g_srv3 and g_srv4.

Example 8-4 `onstat -g cat servers`

```
bash-2.05# onstat -g cat servers
```

GLOBAL-CATALOG CACHE STATISTICS

```
-----  
Current server : Id 100, Nm g_srv1  
Last server slot: (0, 4)  
# free slots : 0  
Broadcast map : <[000d]>  
Leaf server map : <[0014]>  
Root server map : <[0002]>  
Adjacent server map: <[000c]>  
  Id: 100, Nm: g_srv1, Or: 0x0002, off: 0, idle: 500, state Active  
    root Id: 00, forward Id: 00, ishub: TRUE, isleaf: FALSE  
    subtree map: <[001c]>  
  
  Id: 400, Nm: g_srv4, Or: 0x0004, off: 0, idle: 500, state Active  
    root Id: 100, forward Id: 400, ishub: FALSE, isleaf: TRUE  
  
  Id: 200, Nm: g_srv2, Or: 0x0008, off: 0, idle: 500, state Active  
    root Id: 100, forward Id: 200, ishub: TRUE, isleaf: FALSE  
    subtree map: <[0010]>  
  
  Id: 300, Nm: g_srv3, Or: 0x0010, off: 0, idle: 500, state Active  
    root Id: 200, forward Id: 200, ishub: FALSE, isleaf: TRUE
```

In Example 8-4, 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 find out which servers are yet to acknowledge for 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 state is shown as Quiescent then there is a problem with the server definition.

- ▶ isleaf: If set to TRUE it is a leaf server.

Global server replicate statistics

Example 8-5 shows a sample output of replication statistics.

Example 8-5 onstat -g cat repls

```
bash-2.05# onstat -g cat repls

GLOBAL-CATALOG CACHE STATISTICS

REPLICATES
-----
  Parsed statements:
      Id 6553615 table customer

  Inuse databases: stores_demo(1)
      Name: cust_srv1_17_1_customer, Id: 6553615 State: ACTIVE
      Flags: 0x800000 use 0 lastexec Wed Dec 31 16:00:00 1969

      Local Participant: stores_demo:root.customer
      Attributes: TXN scope, all columns sent in updates
      Conflict resolution[Prim::Sec]: [IGNORE]
      Column Mapping: OFF
      Column Name Verification: ON
      No Replicated UDT Columns
```

The following conventions are used in Example 8-5:

- ▶ **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 IDS server
- ▶ **State:** Current state of the replicate. Valid states are FAILED, SUSPENDED, INACTIVE, ACTIVE, DELETED, QUIESCENT, RESUMED. If you see replicate state set to FAILED, the replicate definition is in an inconsistent state.

8.3.2 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, user transactions are blocked until enterprise replication can advance. This situation is called DDRBLOCK mode; it occurs only if the system is severely misconfigured.

Chapter 4 of the IDS manual describes how to configure the logical logs for enterprise replication. It is very 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. In order to understand the logical log requirement and monitor the system, `onstat` provides the log snooping statistics option, `ddr`.

In Example 8-6 we show an example of the `onstat -g ddr` command.

Example 8-6 onstat -g ddr

```
bash-2.05# onstat -g ddr
```

```
IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 1 days
21:59:48 -- 51200 Kbytes
```

```
DDR -- Running --
```

# Event Buffers	Snoopy ID	Snoopy Position	Replay ID	Replay Position	Current ID	Current Position
528	39	317198	39	2ec018	39	318000

```
Log Pages Snooped:
```

From Cache	From Disk	Tossed (LBC full)
1088	704	0

```
Total dynamic log requests: 0
```

```
DDR events queue
```

Type	TX id	Partnum	Row id
------	-------	---------	--------

One of the first things you notice in this example is `DDR -- Running --`. This is normal when the log is configured correctly and ER is operating properly. If there

is a problem in the system due to misconfiguration of the log, you may see Blocked:DDR in the status line as show in Example 8-7.

Example 8-7 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. You will see the onstat output shown in Example 8-7 when the server is in DDRBLOCK mode.

In IDS server versions that 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 log switches. Ideally you should make sure that log switch happens in 60 minute intervals. Another possibility is that the send queue smart blob space (CDR_QDATA_SBSPACE config parameter) is full and the replication threads are waiting for more space in queue smart blob space to enable them to spool replication data to disk.

The following scenario shows the details regarding certain fields from **onstat -g ddr** 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 config parameter is enabled, then ER may request dynamic log addition to avoid a DDRBLOCK situation.

The supported values for CDR_MAX_DYNAMIC_LOGS config parameter are the following:

- 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.

8.3.3 Grouper statistics

Grouper is a subcomponent in ER that receives transactions from Snooper and evaluates the log record for replication. It evaluates the log records, rebuilds the transaction, and queues the transaction for transmission. 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 8-8.

Example 8-8 onstat -g grp

```
bash-2.05# onstat -g grp
```

```
IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 2 days
01:32:45 -- 51200 Kbytes
Grouper at 0xbf69020:
Last Idle Time: (1169793272) 2006/12/25 22:34:32
RSAM interface ring buffer size: 528
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 48
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 597
Log update buffer memory in use: 0
Max log update buffer memory used at once: 95176
Updates from Log: 66188
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
Last Tx to Queuer began : (1169781723) 2006/12/25 19:22:03
Last Tx to Queuer ended : (1169781723) 2006/12/25 19:22:03
Last Tx to Queuer log ID, position: 49,3813784
Open Tx: 0
Serial Tx: 0
Tx not sent: 3
Tx sent to Queuer: 11509
Tx returned from Queuer: 11509
```

```
Events sent to Queuer: 11
Events returned from Queuer: 11
Total rows sent to Queuer: 40129
Open Tx array size: 1024
Table 'customer' at 0xbae7b50 [ ]
```

Some of the useful information elements from Example 8-8 are:

- ▶ Open Tx: Logs are sent to Grouper as soon as they are generated by an active transaction. This number represents the open transactions for which Grouper has not yet received a “commit work” or “rollback work” log record.
- ▶ Serial Tx: Grouper received all log records for these transactions.
- ▶ Tx not sent: The number of transactions that were rejected by Grouper. These transactions have been discarded primarily because there were rolled back.
- ▶ Tx sent to Queuer: Number of transactions that 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

Grouper statistics also produce the information about replicates. For example, Grouper-related statistics can be generated by **onstat -g grp R** as shown in Example 8-9.

Example 8-9 onstat -g grp R

```
bash-2.05# onstat -g grp R
```

```
IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 2 days
01:45:26 -- 51200 Kbytes
Replication Group 13107206 at 0xb958390
  Replication at 0xb9582d8 13107206:13107206 (customer) [FullRowOn ]
    Column Information [ VarUDTs IeeeFloats InOrder Same ]
      In Order: offset 0, size 134
```

If **onstat -g cat repls** shows that 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

Grouper statistics also produce paging statistics. 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, G251-0452. The suboption **onstat -g grp pager** provides current status of the paging system by Grouper. This is shown in Example 8-10.

Example 8-10 onstat -g grp pager

```
bash-2.05# onstat -g grp pager

IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line (LONGTX) -- Up
01:40:51 -- 27648 Kbytes
Blocked:LONGTX
Grouper Pager statistics:
Number of active big transactions: 1
Total number of big transactions processed: 4
```

The output in Example 8-10 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.

8.3.4 Reliable Queue Manager (RQM) statistics

ER in the Informix Dynamic Server maintains two types of data queues, send and receive. Grouper threads running in parallel send completed transactions to the send queue. These transactions are sent to target systems and saved into the receive queue of the target server. This is a place to look for the source of performance or malfunction problems. Onstat provides very detailed information about the two queues. The command used for viewing the queue activity is **onstat -g rqm**. This command is further divided with a number of suboptions, such as **sendq** and **recvq**.

Send Queue statistics

Onstat produces long output as part of **sendq** statistics. Sample output from **onstat -g rqm sendq** is divided into two parts (Example 8-11 and Example 8-11).

Example 8-11 onstat -g rqm sendq (part 1)

IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 22:51:44
-- 5
1200 Kbytes

CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0xbc66020) trg_send

Transaction Spool Name: trg_send_stxn

Insert Stamp: 265/0

Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK

Txns in queue: 18

Log Events in queue: 0

Txns in memory: 18

Txns in spool only: 0

Txns spooled: 0

Unspooled bytes: 3546

Size of Data in queue: 3546 Bytes

Real memory in use: 3546 Bytes

Pending Txn Buffers: 0

Pending Txn Data: 0 Bytes

Max Real memory data used: 10244 (1536000) Bytes

Max Real memory hdrs used: 12272 (1536000) Bytes

Total data queued: 52205 Bytes

Total Txns queued: 265

Total Txns spooled: 0

Total Txns restored: 0

Total Txns recovered: 0

Spool Rows read: 0

Total Txns deleted: 247

Total Txns duplicated: 0

Total Txn Lookups: 1573

Progress Table:

Progress Table is Stable

On-disk table name.....:	spttrg_send
Flush interval (time).....:	30
Time of last flush.....:	1169942922
Flush interval (serial number):	1000
Serial number of last flush...:	2
Current serial number.....:	5

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 there 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 you to configure the system for optimal performance. The following 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 8-12 onstat -g rqm sendq (part 2)

Server	Group	Bytes Queued	Acked	Sent
300	0xc80002	1815	c8/117/277198/0 - c8/117/286198/0	
100	0xc80002	2970	c8/117/270198/0 - c8/117/286198/0	

First Txn (0xbcb41f0) Key: 200/279/0x00271198/0x00000000
Txn Stamp: 248/0, Reference Count: 0.
Txn Flags: Notify
Txn Commit Time: (1169942928) 2006/12/27 16:08:48
Txn Size in Queue: 197
First Buf's (0xbcb43a8) Queue Flags: Resident
First Buf's Buffer Flags: TRG, Stream
NeedAck: Waiting for Acks from <[0002]>
No open handles on txn.

```

Last Txn (0xbc6ebd0) Key: 200/279/0x00286198/0x00000000
Txn Stamp: 265/0, Reference Count: 0.
Txn Flags: Notify
Txn Commit Time: (1169942930) 2006/12/27 16:08:50
Txn Size in Queue: 197
First Buf's (0xbc6ed88) Queue Flags: Resident
First Buf's Buffer Flags: TRG, Stream
NeedAck: Waiting for Acks from <[0012]>

```

```

Tra handle (0xbeac5b8) for thread CDRNr100 at Head_of_Q,Flags: None
Tra handle (0xbead978) for thread CDRNst100 at txn (0xbc6ebd0)
End_of_Q, Flags: None
Tra handle (0xbfb3f60) for thread CDRNr300 at Head_of_Q, Flags: None
Tra handle (0xbfb5320) for thread CDRNst300 at txn (0xbc6ebd0)
End_of_Q, Flags: None
Tra handle (0xc4db020) for thread CDRGeval0 at Head_of_Q,Flags: None
Tra handle (0xbcdelf0) for thread CDRGeval2 at Head_of_Q,Flags: None
Tra handle (0xbcab1f0) for thread CDRACK_0 at Head_of_Q, Flags: None
Tra handle (0xbc981f0) for thread CDRACK_1 at Head_of_Q, Flags:None
Tra handle (0xbc6elf0) for thread CDRGeval1 at Head_of_Q,Flags: None

```

Progress table information

Progress table information is displayed in the format shown in Example 8-13.

Example 8-13 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 sent point or last ack point is unknown you will see either of these transaction keys in the progress table:

```

0xfeeeeeee/0xfeeeeeee/0xfeeeeeee/0xfeeeeeee
0xffffffff/0xffffffff/0xffffffff/0xffffffff

```

- ▶ Transaction key: This is a four part key made up of the following:
 - 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 <[0004]>: 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 **onstat -g cat servers** output. Look for the **Or** from this output, as shown in the following excerpt from Example 8-4 on page 115.

```
Id: 400, Nm: g_srv4, Or: 0x0004, off: 0, idle: 500, state Active
  root Id: 100, forward Id: 400, ishub: FALSE, isleaf: TRUE
Id: 300, Nm: g_srv3, Or: 0x0010, off: 0, idle: 500, state Active
  root Id: 200, forward Id: 200, ishub: FALSE, isleaf: TRUE
```

Receive Queue statistics

The Receive Queue option in **onstat** reports the statistics related to the receive queue. Example 8-14 shows a sample output from **onstat -g rqm recvq**.

Except for a few differences, the receive queue statistics are similar to the send queue statistics. One difference between **sendq** and **recvq** is the following:

- ▶ Sent: Last transaction received from the source server for the given replicate ID.

Except in rare instances, such as replicating very large transactions, you should never see spooling activity in the receive queue. If you do see spooling in the receive queue you should look for the root cause of the problem. Otherwise, you may see increased replication latency.

Example 8-14 *onstat -g rqm recvq*

CDR Reliable Queue Manager (RQM) Statistics:

```
RQM Statistics for Queue (0xae1d020) trg_receive
Transaction Spool Name: trg_receive_stxn
Insert Stamp: 797/0
Communal Stamp: 797/0
Flags: RECV_Q, SPOOLED, PROGRESS_TABLE
Txns in queue:          0
Txns in memory:         0
Txns in spool only:     0
Txns spooled:           0
Unspooled bytes:        0
```

```

Size of Data in queue:      0 Bytes
Real memory in use:        0 Bytes
Pending Txn Buffers:       0
Pending Txn Data:          0 Bytes
Max Real memory data used: 788 (1536000) Bytes
Max Real memory hdrs used  944 (1536000) Bytes
Total data queued:         157009 Bytes
Total Txns queued:         797
Total Txns spooled:        0
Total Txns restored:       0
Total Txns recovered:      0
Spool Rows read:           0
Total Txns deleted:        797
Total Txns duplicated:     0
Total Txn Lookups:         6602
Progress Table:
    Progress Table is Stable
        On-disk table name.....: spttrg_receive
        Not keeping dirty list.
Server Group Bytes Queued  Acked          Sent
-----
200 0xc80002              0 c8/118/122198/0 - c8/118/122198/0
200 0xc80002              0 c8/118/fa198/0   -feee/feee/feee/feee
Traverse handle (0xae251f0) for thread CDRNrA200 at Head_of_Q, Flags:
None
Tra handle (0xb46e820) for thread CDRNr400 at Head_of_Q, Flags: None
Tra handle (0xb51b560) for thread CDRD_2 at Head_of_Q,Flags:Communal
Tra handle (0xb51bf40) for thread CDRD_2 at Head_of_Q,  Flags: None

```

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 8-15 shows a sample output of the control queue statistics generated by **onstat -g rqm cntrlq**.

Example 8-15 onstat -g rqm cntrlq

```

RQM Statistics for Queue (0xbcac020) control_send
Transaction Spool Name: control_send_stxn
Insert Stamp: 20/0
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: 4667 (0) Bytes
Max Real memory hdrs used 1180 (0) Bytes
Total data queued:       16141 Bytes
Total Txns queued:       20
Total Txns spooled:      20
Total Txns restored:     0
Total Txns recovered:    0
Spool Rows read:         0
Total Txns deleted:      20
Total Txns duplicated:   0
Progress Table:
    Progress Table is Stable
        On-disk table name.....: sptcontrol_send
        Flush interval (time).....: 30
        Time of last flush.....: 1169941225
        Flush interval (serial number): 1000
        Serial number of last flush...: 4
        Current serial number.....: 5

```

Server	Group	Bytes	Q	Acked	Sent
300	0x12c0000	0	c8/0/b/0	-	c8/0/b/0
100	0x1	0	c8/0/12/0	-	c8/0/12/0
100	0x1	0	12c/0/2/0	-	12c/0/2/0
100	0	0	c8/0/1/0	-	c8/0/1/0

The control queue is used for replicating, and transferring replication definitions. If you ever see that replication objects (such as replicate, replicate set, and servers) status differs from server to server, this is one of the places to look to find pending control messages.

Example 8-16 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 8-15, are omitted here. In Example 8-16 you can see that 153 bytes of control data is queued for server 200 and also waiting for an acknowledgement.

Example 8-16 onstat -g rqm cntrlq

Server	Group	Bytes Queued	Acked	Sent

200	0x1		153 64/0/e/0	- 64/0/f/0
200	0x1		0 190/0/2/0	- 190/0/2/0
400	0x1900000		0 64/0/5/0	- 64/0/5/0
First Txn (0xb1c1020) Key: 100/0/0x0000000f/0x00000000				
Txn Stamp: 12/0, Reference Count: 0.				
Txn Flags: Spooled				
Txn Commit Time: (1170012928) 2006/12/01/28 11:35:28				
Txn Size in Queue: 185				
First Buf's (0xb1c1bb0) Queue Flags: Spooled, Resident				
First Buf's Buffer Flags: Control				
NeedAck: Waiting for Acks from <[0009]>				
Traverse handle (0xade31f0) for thread CDRNsA200 at txn (0xb1c1020)				
End_of_Q,Flags: None				
Last Txn (0xb1c1020) Key: 100/0/0x0000000f/0x00000000				
Txn Stamp: 12/0, Reference Count: 0.				
Txn Flags: Spooled				
Txn Commit Time: (1170012928) 2006/12/28 11:35:28				
Txn Size in Queue: 185				
First Buf's (0xb1c1bb0) Queue Flags: Spooled, Resident				
First Buf's Buffer Flags: Control				
NeedAck: Waiting for Acks from <[0009]>				
Traverse handle (0xade31f0) for thread CDRNsA200 at txn (0xb1c1020)				
End_of_Q,Flags: None				

8.3.5 Network Interface (NIF) 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 you determine whether or not a site is in a hung state if it is not sending or receiving bytes.

Example 8-17 shows a snapshot of **onstat -g nif** output on server **g_srv1**.

Example 8-17 onstat -g nif

```
IBM Informix Dynamic Server Version 10.00.UC1 -- On-Line -- Up 2 days
12:25:
44 -- 27648 Kbytes

NIF anchor Block: abcbf70
```

```

nifGState      RUN
RetryTimeout    300

```

CDR connections:

Id	Name	State	Version	Sent	Received
400	g_srv4	RUN	8	20084	10056
200	g_srv2	RUN	8	10410	20227

This command lists servers that are directly connected to the current server. The most important information in the **onstat -g nif** 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 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 the large server to improve its 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 8-18 shows sample output from the command **onstat -g nif 200**.

Example 8-18 onstat -g nif 200

```

NIF anchor Block: af58f70
nifGState      RUN
RetryTimeout    300

Detailed Site Instance Block: b477960
siteId          200
siteState        257 <RUN>
siteVersion      8
siteCompress     0
siteNote         0
Send Thread      826 <CDRNsA200>
Recv Thread      827 <CDRnRA200>

```

```

Connection Start      (1170059265) 2006/12/29 00:27:45
Last Send             (1170059559) 2006/12/29 00:32:39
Idle Timeout          30000
Flowblock             Sent 0 Receive 0
NifInfo: b553d60
    Last Updated      (1170059265) 2006/12/29 00:27:45
    State             Connected
    Total sent         1142 (55055 bytes)
    Total recv'd       2443 (319492 bytes)
    Retry              0 (0 attempts)
    Connected          1
Protocol              asf
Proto block           ad44f78
    assoc              aa2dac0
    state              0
    signal             0
Recv Buf              0
Recv Data             0
Send Count            0

```

Example 8-18 shows detailed NIF statistics for server group ID 200. If there is a non-zero value for Flowblock there is a possibility of increased replication latency. This is a potential problem, or the system might 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.

8.3.6 Receive Manager statistics

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 8-19 and Example 8-20.

Example 8-19 onstat -g rcv full (part 1)

```

ServerId: 200
Flags 0x4
ServerId: 400
Flags 0x8
Threads:
    Id Name          State Handle
    822 CDRACK_1     Idle  b477850

```



```

821 CDRACK_0      Idle b553e40
Receive Manager global block b549020
Private
    cdrRM_St_Mutex:          b312ac0
    cdrRM_RInstList:         b477b30
    cdrRM_closeThread:       0
Public
    cdrRM_inst_ct:           2
    cdrRM_State:              00000000
    cdrRM_numSleepers:        4
    cdrRM_DsCreated:          4
    cdrRM_MinDSThreads:       1
    cdrRM_MaxDSThreads:       4
    cdrRM_DSBlock             0
    cdrRM_DSParallelPL        0
    cdrRM_DSFailRate          0.000000
    cdrRM_DSNumRun:           2105
    cdrRM_DSNumLockTimeout    0
    cdrRM_DSNumLockRB         0
    cdrRM_DSNumDeadLocks      0
    cdrRM_DSNumPCommits       0
    cdrRM_ACKwaiting          0
    cdrRM_totSleep:           599
    cdrRM_Sleeptime:          957
    cdrRM_Workload:           0
    cdrRM_optscale:           4
    cdrRM_MinFloatThreads:    2
    cdrRM_MaxFloatThreads:    7
cdrRM_AckThreadCount:        2
    cdrRM_AckWaiters:         2
    cdrRM_AckCreateStamp:     Mon Dec 29 00:27:26 2007
    cdrRM_DSCreateStamp:      Mon Dec 29 00:50:05 2007
    cdrRM_acksInList:         0
    cdrRM_BlobErrorBufs:      0

```

Some of the terms from **onstat -g rcv full** output in Example 8-19 are the following:

- ▶ **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 gotten 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** output shows parallelism statistics and the source server statistics of each as shown in Example 8-20.

Example 8-20 onstat -g rcv full (part 2)

Receive Parallelism Statistics									
Server	Tot.Txn.	Pending	Active	MaxPnd	MaxAct	AvgPnd	AvgAct	CommitRt	
200	2105	0	0	181	3	49.00	1.60	0.78	
Tot Pending:0		Tot Active:0		Avg Pending:49.00		Avg Active:1.60			
Commit Rate:0.78									
Time Spent In RM Parallel Pipeline Levels									
Lev.	TimeInSec	Pcnt.							
0	2685	100.00%							
1	0	0.00%							
2	0	0.00%							
Statistics by Source									
Server	200								
Repl	Txn	Ins	Del	Upd	Last Target	Apply	Last Source		
Commit									
13107205	5	100	0	0	2006/12/29 00:29:11	2006/01/29 00:29:10			
13107202	2100	2100	0	0	2006/12/29 00:50:11	2006/01/29 00:49:55			

The server data displayed in Example 8-20 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, per replicate statistics:

 - Repl: Replicate ID. Get the replicate name and table name from 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.

8.3.7 Data Sync statistics

Data Sync is the subsystem in enterprise replication where transactions are applied to the target table on the target server. The operation is performed by multiple threads in parallel. Statistics about this module can be seen using the **onstat -g dss** command; sample output is shown in Example 8-21.

Example 8-21 onstat -g dss

DS thread statistic					
cmtTime	Tx	Tx	Tx	Last Tx	
Name	< local	Committed	Aborted	Processed	Processed Time

CDRD_2	0	1433	0	1433	(1170064037)
Tables (0.0%):					
Databases: stores_demo					
CDRD_3	0	261	0	261	(1170060611)
Tables (0.0%):					
Databases: stores_demo					
CDRD_4	0	205	0	205	(1170060611)
Tables (0.0%):					
Databases: stores_demo					
CDRD_5	0	202	0	202	(1170060611)
Tables (0.0%):					
Databases: stores_demo					
CDR_DSLOCKWAIT = 5					
CDR_DSCLOSEINTERVAL = 60					

The output in Example 8-21 shows that there are four 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** 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.

8.3.8 A few more troubleshooting tips

In this section we provide you with a few more troubleshooting tips.

Increase in replication latency

There can 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 if you are using a time stamp or stored procedure conflict resolution rule.

If clocks are out of sync, the server will print the following warning message in server message log file:

```
Warning - The operating system time of day clock at the remote site
differs from the local site.
```

This can slow down the data-sync apply performance at the target server. This happens only if time stamp or stored procedure conflict resolution rules are being used.

Moving the ER smartblob queue space

If the server is running out of sbospace defined in enterprise replication configuration, you can either add more space to the sbospace or change smartblob queue space to another sbospace that has sufficient space. If you want to change the name of the sbospace, follow these steps:

1. Replace the old smartblob space name with a new smartblob space name in the onconfig file for the CDR_QDATA_SBSPACE config parameter.
2. Bounce the IDS instance. From this point ER will use the new smartblob space for spooling activity.
3. You can then drop the old smartblob space once it is empty. However, ER may still refer to this smartblob space for retrieving already spooled data, so

wait for space to become empty before you drop it. Never try to drop the old smartblob space using the **-f** (force) option of **onspaces -d** command.

Data not being replicated

In this scenario, a replicate was defined and replication was started, but the data is not getting 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 is showing as active:

- **Cdr list repl**: Shows replicate status in the syscdr database.
- **Onstat -g cat repls**: Shows replicate status in memory.

It is possible that **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.
```

Handling ER queue commands

In this scenario we look at handling ER queue smartblob space, dbspace, and grouper paging smartblob space.

If you are on an IDS version at a level of 10.0x or later, then always create an alarm notification for alarm event class ID 31. This event should be addressed immediately. Event ID 31 notifies you that the ER queue smartblob space, dbspace, or grouper paging sbpace is full and ER is waiting for additional space to be added before it can continue. If additional space is not added, the server may run into DDRBLOCK situation. Later the situation can arise where the log wrap, replay, and snoopy position can be overwritten by the current log position.

If this happens you will see a message, similar to the following message, in the server message log file if ER queue space fills up:

```
16:06:28 CDR QUEUER: Send Queue space is FULL - waiting for space in
sbsp1.
```

You will see a message, similar to the following message, in the server message log file if the ER grouper paging smartblob space fills up:

```
16:06:28 CDR Pager: Paging File full: Waiting for additional space
in sbsp2.
```

Archived

Glossary

Access control list (ACL). The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

Aggregate. Pre-calculated and pre-stored summaries, kept in the data warehouse to improve query performance.

Aggregation. An attribute-level transformation that reduces the level of detail of available data, for example, having a Total Quantity by Category of Items rather than the individual quantity of each item in the category.

Application programming interface. An interface provided by a software product that enables programs to request services.

Asynchronous messaging. A method of communication between programs in which a program places a message on a message queue, and then proceeds with its own processing without waiting for a reply to its message.

Attribute. A field in a dimension table.

BLOB. Binary large object, a block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

Commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

Composite key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

Computer. A device that accepts information (in the form of digitalized data) and manipulates it for some result based on a program or sequence of instructions about how the data is to be processed.

Configuration. The collection of brokers, their execution groups, the message flows and sets that are assigned to them, and the topics and associated access control specifications.

Continuous Data Replication. Refer to Enterprise Replication.

DDL (data definition language). An SQL statement that creates or modifies the structure of a table or database, for example, CREATE TABLE, DROP TABLE, ALTER TABLE, or CREATE DATABASE.

DML (data manipulation language). An INSERT, UPDATE, DELETE, or SELECT SQL statement.

Data append. A data loading technique where new data is added to the database leaving the existing data unaltered.

Data cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content. This is typically to improve the quality, consistency, and usability of the data.

Data federation. The process of enabling data from multiple heterogeneous data sources to appear as though it is contained in a single relational database. Can also be referred to “distributed access.”

Data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope, such as for a department or workgroup. It can be independent, or derived from another data warehouse environment.

Data mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

Data partition. A segment of a database that can be accessed and operated on independently even though it is part of a larger data structure.

Data refresh. A data loading technique where all the data in a database is completely replaced with a new set of data.

Data warehouse. A specialized data environment developed, structured, and used specifically for decision support and informational applications. It is subject oriented rather than application oriented. Data is integrated, non-volatile, and time variant.

Database partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DataBlades. These are program modules that provide extended capabilities for Informix databases and are tightly integrated with the DBMS.

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP® Business Information Warehouse.

Debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

Deploy. Make operational the configuration and topology of the broker domain.

Dimension. Data that further qualifies or describes a measure, or both, such as amounts or durations.

Distributed application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

Dynamic SQL. SQL that is interpreted during execution of the statement.

Engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

Enrichment. The creation of derived data. An attribute-level transformation performed by some type of algorithm to create one or more new (derived) attributes.

Enterprise Replication. An asynchronous, log-based tool for replicating data between IBM Informix Dynamic Server database servers.

Extenders. These are program modules that provide extended capabilities for DB2 and are tightly integrated with DB2.

FACTS. A collection of measures, and the information to interpret those measures in a given context.

Federation. Providing a unified interface to diverse data.

Gateway. A means to access a heterogeneous data source. It can use native access or ODBC technology.

Grain. The fundamental lowest level of data represented in a dimensional fact table.

Instance. A particular realization of a computer process. Relative to the database, the realization of a complete database environment.

Java Database Connectivity. An application programming interface that has the same characteristics as ODBC, but is specifically designed for use by Java database applications.

Java Development Kit. Software package used to write, compile, debug, and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit that enables you to run Java applets and applications.

Materialized query table. A table where the results of a query are stored for later reuse.

Measure. A data item that measures the performance or behavior of business processes.

Message domain. The value that determines how the message is interpreted (parsed).

Message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

Message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible for generating a bit stream for an outgoing message from the internal representation.

Metadata. Typically called data (or information) about data. It describes or defines data elements.

MOLAP. Multidimensional OLAP. Can be called MD-OLAP. It is OLAP that uses a multidimensional database as the underlying data structure.

Multidimensional analysis. Analysis of data along several dimensions, for example, analyzing revenue by product, store, and date.

Multitasking. Operating system capability that allows multiple tasks to run concurrently, taking turns using the resources of the computer.

Multithreading. Operating system capability that enables multiple concurrent users to use the same program. This saves the overhead of initiating the program multiple times.

Nickname. An identifier that is used to reference the object located at the data source that you want to access.

Node group. Group of one or more database partitions.

Node. An instance of a database or database partition.

ODS. (1) Operational data store: A relational table for holding clean data to load into InfoCubes, and can support some query activity. (2) Online Dynamic Server, an older name for IDS.

OLAP. Online analytical processing. Multidimensional data analysis, performed in real time. Not dependent on an underlying data schema.

Open Database Connectivity. A standard application programming interface for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call-level interface (CLI) specification of the X/Open SQL Access Group.

Optimization. The capability to enable a process to execute and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the end user.

Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

Pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

Pivoting. Analysis operation where a user takes a different viewpoint of the results, for example, by changing the way the dimensions are arranged.

Primary key. Field in a table that is uniquely different for each record in the table.

Process. An instance of a program running in a computer.

Program. A specific set of ordered operations for a computer to perform.

Pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be executed. More simply, a pushdown operation is one that is executed at a remote server.

ROLAP. Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

Roll-up. Iterative analysis, exploring facts at a higher level of summarization.

Server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

Static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

Subject area. A logical grouping of data by categories, such as customers or items.

Synchronous messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

Task. The basic unit of programming that an operating system controls. Also see Multitasking.

Thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. Also see Multithreading.

Unit of work. A recoverable sequence of operations performed by an application between two points of consistency.

User mapping. An association made between the federated server user ID and password and the data source (to be accessed) user ID and password.

Virtual database. A federation of multiple heterogeneous relational databases.

Warehouse catalog. A subsystem that stores and manages all the system metadata.

xtree. A query-tree tool that enables you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

ACS	access control system	DCE	distributed computing environment
ADK	Archive Development Kit	DCM	Dynamic Coserver Management
API	application programming interface	DCOM	Distributed Component Object Model
AQR	automatic query rewrite	DDL	data definition language
AR	access register	DES	Data Encryption Standard
ARM	automatic restart manager	DIMID	Dimension Identifier
ART	access register translation	DLL	dynamic link library
ASCII	American Standard Code for Information Interchange	DML	data manipulation language
AST	application summary table	DMS	database managed space
BLOB	binary large object	DPF	data partitioning facility
BW	Business Information Warehouse (SAP)	DRDA®	Distributed Relational Database Architecture™
CCMS	Computing Center Management System	DSA	Dynamic Scalable Architecture
CDR	Continuous Data Replication	DSN	data source name
CFG	Configuration	DSS	decision support system
CLI	call-level interface	EAI	Enterprise Application Integration
CLOB	character large object	EBCDIC	Extended Binary Coded Decimal Interchange Code
CLP	command line processor	EDA	enterprise data architecture
CORBA	Common Object Request Broker Architecture	EDU	engine dispatchable unit
CPU	central processing unit	EGM	Enterprise Gateway Manager
CS	Cursor Stability	EJB™	Enterprise Java Beans
DAS	DB2 Administration Server	ER	Enterprise Replication
DB	database	ERP	Enterprise Resource Planning
DB2 II	DB2 Information Integrator	ESE	Enterprise Server Edition
DB2 UDB	DB2 Universal Database™	ETL	Extract, Transform, and Load
DBA	database administrator	FP	fix pack
DBM	database manager	FTP	File Transfer Protocol
DBMS	database management system	Gb	gigabits

GB	gigabytes	LPAR	logical partition
GUI	graphical user interface	LV	logical volume
HADR	High Availability Disaster Recovery	Mb	megabits
HDR	High Availability Data Replication	MB	megabytes
HPL	High Performance Loader	MDC	multidimensional clustering
I/O	input/output	MPP	massively parallel processing
IBM	International Business Machines Corporation	MQI	message queuing interface
ID	identifier	MQT	materialized query table
IDE	Integrated Development Environment	MRM	message repository manager
IDS	Informix Dynamic Server	MTK	DB2 Migration Toolkit for Informix
II	Information Integrator	NPI	non-partitioning index
IMS™	Information Management System	ODBC	Open Database Connectivity
ISAM	Indexed Sequential Access Method	ODS	operational data store
ISM	Informix Storage Manager	OLAP	online analytical processing
ISV	independent software vendor	OLE	object linking and embedding
IT	information technology	OLTP	online transaction processing
ITR	internal throughput rate	ORDBMS	Object Relational Database Management System
ITSO	International Technical Support Organization	OS	operating system
IX	index	O/S	operating system
J2EE™	Java 2 Platform Enterprise Edition	PDS	partitioned data set
JAR	Java Archive	PIB	parallel index build
JDBC™	Java Database Connectivity	PSA	persistent staging area
JDK™	Java Development Kit	RBA	relative byte address
JE	Java Edition	RBW	red brick warehouse
JMS	Java Message Service	RDBMS	Relational Database Management System
JRE™	Java Runtime Environment	RID	record identifier
JVM™	Java Virtual Machine	RR	repeatable read
KB	kilobyte (1024 bytes)	RS	read stability
LDAP	Lightweight Directory Access Protocol	SCB	session control block
		SDK	Software Developers Kit
		SID	surrogate identifier
		SMIT	Systems Management Interface Tool
		SMP	symmetric multiprocessing

SMS	System Managed Space
SOA	service oriented architecture
SOAP	Simple Object Access Protocol
SPL	Stored Procedure Language
SQL	structured query
TCB	thread control block
TMU	table management utility
TS	table space
UDB	Universal Database
UDF	user-defined function
UDR	user-defined routine
URL	Uniform Resource Locator
VG	volume group (RAID disk terminology).
VLDB	very large database
VP	virtual processor
VSAM	virtual sequential access method
VTI	virtual table interface
WSDL	Web Services Definition Language
WWW	World Wide Web
XBSA	X-Open Backup and Restore APIs
XML	Extensible Markup Language
XPS	Informix Extended Parallel Server

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

For information about ordering these publications, see “How to get Redbooks”. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-7299

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Informix Dynamic Server Administrator's Guide*, G251-2267-02
- ▶ *IBM Informix Enterprise Replication Guide*, G251-1254

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

A

Aborted Transaction Spooling 71
administration 19
administration of replication objects 71
AIX 8–9, 11, 66
application performance 22
architecture 8
artificial primary key 68
asynchronous 5, 11, 21, 23, 33, 112
asynchronous HDR 5, 23
asynchronous I/O 11
asynchronous mode 21, 23
ATS *See* Aborted Transaction Spooling

B

backing up instances 26
backup 2, 4, 14–15, 17, 26, 42, 62
Banda 8, 10, 16, 66, 92
BLOB 20
Block 128–129
bootinfo 8
BUFFERS 12

C

CA 2–3
 see Continuous Availability
Cache 117
Cascade topology 44
checkpoint duration 12
cluster environment 39
commit 5, 22, 49–50, 121, 125, 133
communications connections 64
communications network 62
component failure 2, 5
composite primary key 68
configuration 8–9, 11, 13, 20–21, 24, 27, 43, 45, 70
configuration parameters 12
conflict options 99
conflict resolution 63–64, 68, 70, 75, 119, 134
conflicting updates 63
consolidation 57–58
Continuous Availability 2

control messages 112
control queue 126
control queue statistics 126

D

data consolidation 57
Data Definition Language 20
data distribution 57–58
data integrity 28
data manipulation language 20
data movement 50, 53
data partitioning 47
data queues 121
data replication 4, 13, 109
data replication buffer 13
data sync 133
data sync threads 134
data transparency 51
data types 89
data warehousing 44
database configuration 11
database instance 4, 9, 22
database objects 4
database server 4–5, 8, 13–14, 17, 24, 40, 47, 49, 53, 56, 59, 61, 86, 112
DataBlade 3
dbaccessdemo 66
DBMS 64, 68, 89
DBPATH 33–34
DBSERVERALIAS 24–25, 35
DBSERVERNAME 24–25, 35, 71
dbspace 8, 25, 70, 86, 135
DDL - *see* Data Definition Language
DDRBLOCK mode 117
demo_on 10
deterministic algorithm 63
disaster 20
disaster recovery 43
disaster tolerance 21
disk 12, 15–16
DML - *see* Data Manipulation Language
DRAUTO 11, 13, 24–25, 27–33
DRINTERVAL 11, 13, 21, 23–25, 33

DRLOSTFOUND 13, 25
DROP 69
DROP TABLE 69
DRTIMEOUT 11, 13, 22, 25
DSA - *see* Dynamic Scalable Architecture
Dynamic Scalable Architecture 3

E

EBR - *see* external backup and restore
Enterprise Replication 3
enterprise replication 8
ER 8–9
ER - *see* Enterprise Replication
ER queue commands 135
exclusive set 82, 102, 104
external backup and restore 16

F

failover 3–4, 12, 29, 35
failure 2, 5, 20, 22–24, 27, 33–34, 135
fast recovery 21
Fault Tolerant 2
Flowblock 130
FO *see* Failover
forest 59–60, 62–63
forest of trees 62
forest of trees topology 60
frequency options 99
FT *see* Fault Tolerant
fully-connected topology 59

G

group statistics 119–120
groupers 110

H

hardware cluster 37–40, 43
HDR pair 20, 23, 42, 45, 47, 62
 instantiating 23
heartbeat monitor 23
hierarchical configuration 59–60, 62–63, 67
hierarchical tree 60
hierarchical tree topology 59
high availability 7, 19
High Availability Data Replication 7
high performance 109
historical data 123

hot backup 4, 40
HP-UX 8

I

IDS instance 44, 48, 56, 66, 69
INACTIVE 116
Informix Server Administrator 71
INFORMIXSERVER 10, 14, 25, 33–35
INFORMIXSQLHOSTS 10, 25
INSERT 57, 69, 76
instance 14, 17, 20–22, 25, 34, 46, 56, 67, 72, 75,
83, 86, 97, 105
 creation 20
 operation 27, 72
 shared memory 20, 24, 34–35
Instantiating an HDR pair 23
ISA - *see* Informix Server Administrator
isleaf 115

J

Java 9

K

Kodiak 8, 10–11, 15–16

L

latency 121, 125, 129–130, 134
leaf node 8–9, 59, 75
Linux 9, 66
list replicates 97, 101, 103
list replicateset 101
load 10, 21, 42, 68, 123
log buffers 21
logical consistency 23
logical log buffer 12
logical log file 125
logical logs 12, 15, 20–21, 26–27, 30–33, 117
logical restore 30–31, 41
logs 14–15, 120

M

machine clock synchronization 134
mastered replicate 84
max real memory 123
memory 12, 34–35, 63, 70, 121–123, 127, 135
metadata 84, 111–112
mirrored environment 20

mirroring 20, 30
mode 99
model 47
modify replicate 99
modifying replicates 99
multiple instances 43

N

network facilities 62
network failure 32
network interconnect 22
network interface statistics 128–129
network-oriented issues 28
nif - see network interface statistics
Nile 8–9, 66, 71
non-exclusive set 102
non-root servers 59–60

O

OLTP 42, 44
onbar 14–16, 26
ONCONFIG 10–11, 21–22, 24–25
onconfig 11, 70–71, 119, 134
ONCONFIG file 25
one-way propagation 75
online duplicate hardware 2
online.log file 17
onlog utility 33
onmode 17, 24–26, 29–33, 41
onspaces 20, 25, 70, 135
onstat 10, 16–17, 26
ontape 14–15, 17, 26–27, 42
ontape utility 15
Opaque 89
opaque UDT 89

P

page 64
Pages 117
parallelism 111, 132
partitioning 46–47, 50–51, 53
partitioning data 48
pending txn buffers 123
performance 109
performance tuning 12
physical failure 27, 30, 32
physical restore 26, 30–31, 41

primary 8, 12–17, 19–26, 28, 34, 58, 62, 105
primary key 68, 76, 87–88
primary target 58
processes 21, 80, 89
progress tables 111

Q

queue commands 135
Queue Manager 121–122
Queue Spool Areas 95
queues 95, 112, 121
quiescent 115

R

RAID 2–3, 20, 30
real time 27
realizing a template 86
Receive Manager 130
Receive Manager statistics 130
recovery 13, 21, 27–28
referential integrity 105
replicas 55, 87
replicate 22, 56–58, 61, 64, 66, 71, 75–76, 78, 80–86, 91–92, 95, 97–102, 104–105, 109–114, 116, 120, 124–127, 133, 135
replicate definitions 75, 82
replicate group 57
replicate set 57, 66, 81–83, 92, 101–102, 104–105, 127
replicates 56–58, 64–65, 67–69, 75–76, 78–79, 81–84, 87, 89, 91, 97–98, 101–102, 104–105, 120
replication 13, 21–24, 28, 33, 91–93, 95–96, 102, 104–106
replication environment 92, 109
replication latency 134
replication network 64
replication scheme 59
replication server 71, 89, 92, 94, 96, 105
replication topologies 59
replication traffic 22
resolution scheme 70
restore 4, 16, 24, 41
restoring instances 26
RIS - see Row Information Spooling
roles 76
roll forward 21, 27, 41
rollback 5, 22, 121
root database server 59

root server 72
Row Information Spooling 71, 94

S

scalability 51
scheduled maintenance 29
schema 44, 47–48, 50–53
scope options 99
secondary 8, 12–17, 19–25, 34, 62
security 42
SELECT 57, 76, 99–100
serial data type 68
server 64, 91
server group 34–35, 49, 93–95, 130
SET 101, 103
shadow columns 64, 68
shared memory 20
sharing 58, 103
Single Point of Failure 2
smartblob 135
smartblob queue space 134
snooper 110, 119
Solaris cluster 40
split transaction 105
SPoF
 see Single Point of Failure 2
SQL 4, 28, 33–34, 48, 65, 70
sqlhosts file 34–35, 64, 66, 71
Star topology 44
starting the replicate set 86
STDIO 15–16, 27
stored procedures 51, 64
SUN Solaris 8
suspended 115
sync server 86
synchronous 21–23
synchronous mode 23
synonyms 47–48, 51

T

table level data partitioning 47–48
table partitioning 50
tables 6, 47, 50–51, 57, 68–69, 80–81, 83–84, 86–87, 103, 106, 110, 116, 120
TAPEDEV 15
TCP/IP 112
template 66, 83–88, 101, 105–107, 116
test environment 8–9

topology planning 96
total data queued 123
total txns queued 123
total txns spooled 123
traffic - replication 22
transition 29
triggers 51
trusted connection 64
tuning 12, 97
txns in memory 123
txns in queue 123
txns in spool only 123
txns spooled 123
typed tables 69

U

UDR 89
 also see user-defined routines
UDT 89, 116
 also see user-defined types
uni-directional replicates 76
update-anywhere 58, 75–76
update-anywhere replication 63
updating 63
user-defined routines 3, 89
user-defined types 65, 89

V

views 49, 51–52
Virtual Processor 3
VP 3, 70
 also see Virtual Processor

W

workload considerations 63
workload distribution 58
workload sharing 58



Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution



Redbooks

**High availability to
reduce or eliminate
costly downtime**

**Enterprise
replication for fast
data distribution**

**Reduced deployment
and management
costs**

This book provides an overview of the high availability and enterprise replication features of Informix Dynamic Server (IDS) Version 10, which can simplify and automate enterprise database deployment. Version 10 offers patent-pending technology that virtually eliminates downtime and automates many of the tasks associated with deploying mission-critical enterprise systems. The high availability capability can reduce or eliminate costly downtime through, as examples, the introduction of online index rebuild, the capability to create and drop indexes online without locking tables, and point-in-time table-level restore. Enhanced enterprise replication provides solutions for those customers requiring reliable and quick dissemination of data across a global organization. The replicated data can also participate in the high availability environment with capabilities such as seamless online resynchronization of enterprise replication nodes at startup, for example. There is also enhanced capability for high-availability disaster recovery customers in the form of the ability to resend primary server indexes to HDR secondary servers without requiring a rebuild of the index on the primary server. These significant and powerful capabilities can enable you to intelligently replicate (distribute) your enterprise data to improve the availability of that data to better service the requirements of your enterprise.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks