

IBM Informix Dynamic Server Administrator's Guide

Version 9.4
March 2003
Part No. CT1UCNA (Volume 1) and CT1SXNA (Volume 2)

Note:

Before using this information and the product it supports, read the information in the appendix entitled "Notices."

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 2003. All rights reserved.

US Government User Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

List of Chapters

Section I The Database Server

Chapter 1 Installing and Configuring the Database Server

Chapter 2 Configuration Parameters

Chapter 3 Client/Server Communications

Chapter 4 Initializing the Database Server

Section II Disk, Memory, and Process Management

Chapter 5 Virtual Processors and Threads

Chapter 6 Managing Virtual Processors

Chapter 7 Shared Memory

Chapter 8 Managing Shared Memory

Chapter 9 Data Storage

Chapter 10 Managing Disk Space

Section III Logging and Log Administration

Chapter 11 Logging

Chapter 12 Managing Database-Logging Mode

Chapter 13 Logical Log

Chapter 14 Managing Logical-Log Files

Chapter 15 Physical Logging, Checkpoints, and Fast Recovery

Chapter 16 Managing the Physical Log

Section IV Fault Tolerance

Chapter 17 Mirroring

Chapter 18 Using Mirroring

Chapter 19 High-Availability Data Replication

Chapter 20 Using High-Availability Data Replication

Chapter 21 Consistency Checking

Section V Distributed Data

Chapter 22 Multiphase Commit Protocols

Chapter 23 Recovering Manually from Failed Two-Phase Commit

Table of Contents

Introduction

In This Introduction	3
About This Manual	3
Types of Users	3
Software Dependencies	4
Assumptions About Your Locale.	4
Demonstration Database	5
New Features in Dynamic Server, Version 9.4	6
Database Server Usability Enhancements.	6
Security Enhancements	7
Features from Dynamic Server 9.3	7
Features from Dynamic Server 9.21	9
Organizational Changes to This Manual Since Version 9.2	9
Documentation Conventions	10
Typographical Conventions	10
Icon Conventions	11
Sample-Code Conventions.	12
Additional Documentation	13
Related Reading	15
Compliance with Industry Standards	16
IBM Welcomes Your Comments	16

Section I The Database Server

Chapter 1 Installing and Configuring the Database Server

In This Chapter	1-5
Planning for the Database Server	1-6
Considering Your Priorities	1-6
Considering Your Environment	1-7
Configuring the Operating System	1-7
Configuring Windows Memory	1-8
Modifying UNIX Kernel Parameters	1-8
Allocating Disk Space	1-8
Using Large Chunks	1-9
Creating Chunk Files on UNIX	1-9
Providing NTFS Partitions in Windows	1-10
Setting Permissions, Ownership, and Group	1-10
Creating Standard Device Names	1-11
Setting Environment Variables	1-12
Setting GLS Environment Variables	1-13
Setting Environment Variables on UNIX.	1-14
Setting Environment Variables on Windows	1-15
Configuring Connectivity	1-16
The sqlhosts File on UNIX	1-16
The sqlhosts Registry on Windows.	1-17
Configuring Connectivity Using ISA	1-17
Configuring the Database Server	1-18
Preparing the ONCONFIG Configuration File	1-18
Using Server Setup in ISA to Customize Your Configuration.	1-20
Using IBM Informix Server Administrator to Update the ONCONFIG File	1-20
Using the Instance Manager to Create a New Database Server Instance	1-21
Configuring Java Support.	1-21
Starting and Administering the Database Server	1-22
Starting the Database Server and Initializing Disk Space	1-22
Preparing for Automatic Startup	1-23
Preparing to Connect to Applications	1-25
Creating Storage Spaces and Chunks	1-25
Supporting Large Chunks.	1-26
Setting Up Your Backup System and Storage	1-26

Performing Routine Administrative Tasks	1-27
Changing Database Server Modes	1-28
Backing Up Data and Logical-Log Files	1-28
Monitoring Database Server Activity	1-28
Checking for Consistency	1-28
Performing Additional Administrative Tasks	1-29
Using Mirroring	1-29
Managing Database-Logging Status.	1-29
Managing the Logical Log	1-30
Managing the Physical Log.	1-30
Managing Shared Memory	1-31
Managing Virtual Processors	1-31
Managing Parallel Database Query	1-32
Using Data Replication	1-32
Using Auditing	1-33
Using Distributed Queries	1-33
Monitoring Database Server Activity	1-34
Event Alarms	1-34
IBM Informix Server Administrator (ISA).	1-35
Message Log.	1-35
ON-Monitor	1-36
oncheck Utility	1-36
onperf Tool	1-36
onstat Utility.	1-37
SMI Tables	1-37
System Console.	1-37
UNIX Operating-System Tools	1-38
Windows Event Viewer	1-38
Windows Performance Monitor	1-39
Windows Utilities	1-40

Chapter 2 **Configuration Parameters**

In This Chapter	2-3
Database Server Identification Parameters	2-3
Disk-Space Parameters	2-4
Root Dbspace	2-4
Mirror of Root Dbspace	2-5
Other Space-Management Parameters	2-5

Logging Parameters	2-6
Logical Log.	2-6
Physical Log Parameters	2-7
Backup and Restore Parameters.	2-7
Message-Log Parameters	2-8
Shared-Memory Parameters	2-9
Shared-Memory Size Allocation	2-9
Shared-Memory Space Allocation	2-10
Shared-Memory Buffer Control	2-11
SQL Statement Cache Usage	2-11
Decision-Support Parameters	2-12
Database Server Process Parameters	2-13
Virtual Processor Parameters.	2-13
Time Intervals	2-14
Restore Parameters	2-14
High-Availability Data-Replication Parameters	2-15
Event-Alarm Parameters	2-15
Dump Parameters	2-16
Specialized Parameters	2-16
Auditing Parameters	2-17
Optical Media Parameters.	2-17
UNIX Parameters	2-17
Monitoring Configuration Information	2-18

Chapter 3

Client/Server Communications

In This Chapter	3-3
Client/Server Architecture	3-3
Network Protocol	3-4
Network Programming Interface	3-5
Windows Network Domain	3-5
Database Server Connection	3-6
Multiplexed Connection	3-7
Connections That the Database Server Supports	3-8
Local Connections	3-10
Shared-Memory Connections	3-10
Stream-Pipe Connections	3-11
Named-Pipe Connections	3-11
Local-Loopback Connections	3-12
Communication Support Services	3-12

Connectivity Files	3-13
Network-Configuration Files	3-13
Network Security Files	3-17
Communication Support Modules (CSMs)	3-20
The sqlhosts File and the SQLHOSTS Registry Key	3-29
The sqlhosts Information	3-32
Connectivity Information	3-34
Group Information	3-48
Alternatives for TCP/IP Connections	3-48
ONCONFIG Parameters for Connectivity	3-53
DBSERVERNAME Configuration Parameter.	3-54
DBSERVERALIASES Configuration Parameter	3-54
NETTYPE Configuration Parameter.	3-55
Environment Variables for Network Connections	3-56
Examples of Client/Server Configurations	3-56
Using a Shared-Memory Connection	3-57
Using a Local-Loopback Connection	3-58
Using a Network Connection	3-59
Using Multiple Connection Types	3-60
Accessing Multiple Database Servers	3-62
Using IBM Informix MaxConnect	3-63

Chapter 4 Initializing the Database Server

In This Chapter	4-3
Types of Initialization	4-3
Initializing Disk Space	4-4
Initialization Steps	4-5
Process Configuration File	4-6
Create Shared-Memory Portions	4-7
Initialize Shared-Memory	4-8
Initialize Disk Space	4-8
Start All Required Virtual Processors	4-8
Make Necessary Conversions	4-9
Initiate Fast Recovery.	4-9
Initiate a Checkpoint	4-9
Document Configuration Changes	4-9
Create the oncfg_servername.servernum File	4-10
Drop Temporary Tblspaces.	4-10
Set Forced Residency If Specified.	4-10
Return Control to User	4-11

Create sysmaster Database and Prepare SMI Tables	4-11
Create the sysutils Database	4-12
Monitor Maximum Number of User Connections	4-12
Database Server Operating Modes	4-12
Changing Database Server Operating Modes	4-14
Users Permitted to Change Modes.	4-14
ISA Options for Changing Modes	4-15
On-Monitor Options for Changing Modes	4-16
Command-Line Options for Changing Modes.	4-16

Section II Disk, Memory, and Process Management

Chapter 5 Virtual Processors and Threads

In This Chapter	5-3
Virtual Processors	5-3
Threads	5-4
Types of Virtual Processors	5-5
Advantages of Virtual Processors	5-9
How Virtual Processors Service Threads	5-12
Control Structures	5-13
Context Switching	5-13
Stacks.	5-15
Queues	5-16
Mutexes	5-19
Virtual-Processor Classes	5-19
CPU Virtual Processors.	5-19
User-Defined Classes of Virtual Processors	5-24
Java Virtual Processors	5-26
Disk I/O Virtual Processors	5-27
Network Virtual Processors	5-31
Communications Support Module Virtual Processor	5-38
Optical Virtual Processor	5-38
Audit Virtual Processor	5-38
Miscellaneous Virtual Processor	5-38

Chapter 6 Managing Virtual Processors

In This Chapter	6-3
Setting Virtual-Processor Configuration Parameters	6-3
Setting Virtual-Processor Parameters with a Text Editor	6-4
Setting Virtual-Processor Parameters with ISA	6-5
Setting Virtual-Processor Parameters with ON-Monitor	6-6
Starting and Stopping Virtual Processors	6-6
Adding Virtual Processors in Online Mode	6-7
Dropping CPU and User-Defined Virtual Processors	6-8
Monitoring Virtual Processors	6-9
Monitoring Virtual Processors with Command-Line Utilities	6-9
Monitoring Virtual Processors with SMI Tables	6-12

Chapter 7 Shared Memory

In This Chapter	7-5
Shared Memory	7-5
Shared-Memory Use	7-6
Shared-Memory Allocation.	7-7
Shared-Memory Size	7-9
Action to Take If SHMTOTAL Is Exceeded	7-10
Processes That Attach to Shared Memory	7-10
How a Client Attaches to the Communications Portion	7-11
How Utilities Attach to Shared Memory	7-11
How Virtual Processors Attach to Shared Memory.	7-12
Resident Shared-Memory Segments	7-16
Resident Portion of Shared Memory	7-16
Shared-Memory Header.	7-17
Shared-Memory Buffer Pool	7-17
Logical-Log Buffer.	7-19
Physical-Log Buffer	7-20
High-Availability Data-Replication Buffer	7-21
Lock Table	7-21
Virtual Portion of Shared Memory	7-22
Management of the Virtual Portion of Shared Memory	7-23
Components of the Virtual Portion of Shared Memory	7-24
Data-Distribution Cache	7-30
Communications Portion of Shared Memory	7-32
Virtual-Extension Portion of Shared Memory	7-32

Concurrency Control	7-33
Shared-Memory Mutexes	7-33
Shared-Memory Buffer Locks	7-33
Database Server Thread Access to Shared Buffers	7-34
LRU Queues	7-34
Configuring the Database Server to Read Ahead	7-39
Database Server Thread Access to Buffer Pages	7-40
Flushing Data to Disk	7-40
Flushing Buffer-Pool Buffers	7-41
Flushing Before-Images First	7-41
Flushing the Physical-Log Buffer	7-42
Synchronizing Buffer Flushing	7-42
Describing Flushing Activity	7-43
Flushing the Logical-Log Buffer	7-45
Buffering Large-Object Data	7-46
Writing Simple Large Objects	7-47
Accessing Smart Large Objects	7-49
Memory Use on 64-Bit Platforms	7-50

Chapter 8 **Managing Shared Memory**

In This Chapter	8-3
Setting Operating-System Shared-Memory Configuration Parameters	8-3
Maximum Shared-Memory Segment Size	8-4
Shared-Memory Lower-Boundary Address	8-5
Semaphores	8-6
Setting Database Server Shared-Memory Configuration Parameters	8-6
Setting Parameters for Resident Shared Memory	8-7
Setting Parameters for Virtual Shared Memory	8-8
Setting Parameters for Shared-Memory Performance	8-9
Setting Shared-Memory Parameters with a Text Editor	8-9
Setting Shared-Memory Parameters with ISA	8-10
Setting Shared-Memory Parameters with ON-Monitor	8-10
Setting SQL Statement Cache Parameters	8-11
Reinitializing Shared Memory	8-12
Turning Residency On or Off for Resident Shared Memory	8-12
Turning Residency On or Off in Online Mode	8-13
Turning Residency On or Off When Restarting the Database Server	8-13
Adding a Segment to the Virtual Portion of Shared Memory	8-14

Monitoring Shared Memory	8-14
Monitoring Shared-Memory Segments	8-14
Monitoring the Shared-Memory Profile and Latches	8-15
Monitoring Buffers	8-17
Monitoring Buffer-Pool Activity	8-20

Chapter 9

Data Storage

In This Chapter	9-5
Physical and Logical Units of Storage	9-5
Chunks	9-6
Disk Allocation for Chunks.	9-7
Offsets	9-9
Pages	9-10
Blobpages	9-11
Sbpages	9-12
Extents	9-14
Dbspaces	9-16
Control of Where Data Is Stored	9-16
Root Dbspace	9-18
Temporary Dbspaces	9-19
Blobspaces	9-20
Sbspaces	9-21
Advantages of Using Sbspaces	9-21
Sbspaces and Enterprise Replication	9-22
Metadata, User Data, and Reserved Area	9-22
Control of Where Data Is Stored	9-23
Storage Characteristics of Sbspaces	9-25
Levels of Inheritance for Sbspace Characteristics	9-28
More Information About Sbspaces	9-30
Temporary Sbspaces	9-32
Comparison of Temporary and Standard Sbspaces.	9-33
Temporary Smart Large Objects	9-34
Extspaces	9-35
Databases	9-35
Tables	9-36
Table Types for Dynamic Server	9-38
Standard Permanent Tables	9-39
RAW Tables	9-39
Temp Tables	9-40

Properties of Table Types	9-40
Temporary Tables	9-42
Tblspaces	9-45
Maximum Number of Tblspaces in a Table	9-46
Table and Index Tblspaces	9-46
Extent Interleaving	9-48
Table Fragmentation and Data Storage	9-49
Amount of Disk Space Needed to Store Data	9-49
Size of the Root Dbspace	9-49
Amount of Space That Databases Require	9-52
Disk-Layout Guidelines	9-52
Dbspace and Chunk Guidelines.	9-53
Table-Location Guidelines	9-54
Sample Disk Layouts	9-55
Logical-Volume Manager	9-60

Chapter 10 **Managing Disk Space**

In This Chapter	10-5
Allocating Disk Space	10-6
Specifying an Offset.	10-6
Allocating Cooked File Spaces on UNIX.	10-8
Allocating Raw Disk Space on UNIX	10-9
Creating Symbolic Links to Raw Devices	10-10
Allocating NTFS File Space on Windows	10-10
Allocating Raw Disk Space on Windows	10-11
Specifying Names for Storage Spaces and Chunks	10-12
Specifying the Maximum Size of Chunks	10-13
Specifying the Maximum Number of Chunks and Storage Spaces	10-13
Backing Up After You Change the Physical Schema	10-13
Managing Dbspaces	10-14
Creating a Dbspace	10-14
Creating a Temporary Dbspace	10-16
What to Do If You Run Out of Disk Space	10-17
Adding a Chunk to a Dbspace or Blobospace	10-17
Managing Blobspaces	10-19
Creating a Blobospace	10-19
Preparing Blobspaces to Store TEXT and BYTE Data	10-21
Determining Blobpage Size	10-21

Managing Sbspaces	10-23
Creating an Sbspace	10-23
Sizing Sbspace Metadata	10-25
Adding a Chunk to an Sbspace	10-26
Altering Storage Characteristics of Smart Large Objects	10-27
Creating a Temporary Sbspace	10-27
Dropping a Chunk	10-29
Verifying Whether a Chunk Is Empty	10-29
Dropping a Chunk from a Dbspace with onspaces	10-30
Dropping a Chunk from a Blobspace	10-30
Dropping a Chunk from an Sbspace with onspaces	10-30
Dropping a Storage Space	10-31
Preparing to Drop a Storage Space	10-32
Dropping a Mirrored Storage Space	10-32
Dropping a Storage Space with onspaces	10-32
Dropping a Dbspace or Blobspace with ON-Monitor	10-33
Backing Up After Dropping a Storage Space	10-33
Managing Extspaces	10-34
Creating an Extspace	10-34
Dropping an Extspace	10-35
Skipping Inaccessible Fragments	10-35
Using the DATASKIP Configuration Parameter	10-35
Using the Dataskip Feature of onspaces	10-36
Using onstat to Check Dataskip Status	10-36
Using the SQL Statement SET DATASKIP	10-36
Effect of the Dataskip Feature on Transactions	10-37
Determining When to Use Dataskip	10-38
Monitoring Fragmentation Use	10-39
Displaying Databases	10-40
Using SMI Tables	10-40
Using ISA	10-40
Using ON-Monitor	10-40
Monitoring Disk Usage	10-41
Monitoring Chunks	10-41
Monitoring Tblspaces and Extents	10-48
Monitoring Simple Large Objects in a Blobspace	10-49
Monitoring Sbspaces	10-53
Loading Data Into a Table	10-61

Section III Logging and Log Administration

Chapter 11 Logging

In This Chapter	11-3
Database Server Processes That Require Logging	11-3
Transaction Logging	11-6
Logging of SQL Statements and Database Server Activity	11-6
Activity That is Always Logged.	11-7
Activity Logged for Databases with Transaction Logging	11-9
Activity That is Not Logged	11-10
Database-Logging Status	11-10
Unbuffered Transaction Logging	11-11
Buffered Transaction Logging	11-12
ANSI-Compliant Transaction Logging	11-12
No Database Logging	11-12
Databases with Different Log-Buffering Status	11-13
Database Logging in an X/Open DTP Environment.	11-13
Settings or Changes for Logging Status or Mode	11-14

Chapter 12 Managing Database-Logging Mode

In This Chapter	12-3
Changing Database-Logging Mode	12-4
Modifying Database-Logging Mode with ondblog	12-5
Changing Buffering Mode with ondblog.	12-5
Canceling a Logging Mode Change with ondblog	12-6
Ending Logging with ondblog	12-6
Making a Database ANSI Compliant with ondblog	12-6
Changing the Logging Mode of an ANSI-Compliant Database	12-7
Modifying Database Logging Mode with ontape	12-7
Turning On Transaction Logging with ontape	12-7
Ending Logging with ontape.	12-8
Changing Buffering Mode with ontape	12-8
Making a Database ANSI Compliant with ontape	12-8
Modifying Database Logging Mode with ISA	12-9
Modifying Database Logging Mode with ON-Monitor	12-9
Modifying the Table-Logging Mode	12-10
Altering a Table to Turn Off Logging	12-10
Altering a Table to Turn On Logging	12-10
Monitoring Transactions	12-11

Monitoring the Logging Mode of a Database	12-11
Monitoring the Logging Mode with SMI Tables.	12-11
Monitoring the Logging Mode with ON-Monitor	12-12
Monitoring the Logging Mode with ISA	12-12

Chapter 13 **Logical Log**

In This Chapter	13-3
What Is the Logical Log?	13-3
Location of Logical-Log Files	13-4
Identification of Logical-Log Files	13-5
Status Flags of Logical-Log Files	13-6
Size of the Logical Log	13-7
Number of Logical-Log Files	13-7
Performance Considerations	13-8
Dynamic Log Allocation	13-9
Freeing of Logical-Log Files	13-10
Action If the Next Logical-Log File Is Not Free	13-10
Action if the Next Log File Contains the Last Checkpoint	13-11
Logging Blobspaces and Simple Large Objects	13-11
Switching Log Files to Activate Blobspaces	13-12
Backing Up Log Files to Free Blobpages	13-12
Backing Up Blobspaces After Inserting or Deleting TEXT and BYTE Data	13-13
Logging Sbspaces and Smart Large Objects	13-13
Using Sbspace Logging	13-13
Using Smart-Large-Object Log Records	13-16
Preventing Long Transactions When Logging Smart-Large-Object Data	13-16
Logging Process	13-17
Dbspace Logging	13-17
Blobpace Logging.	13-17

Chapter 14 **Managing Logical-Log Files**

In This Chapter	14-3
Estimating the Size and Number of Log Files	14-4
Estimating the Log Size When Logging Smart Large Objects	14-5
Estimating the Number of Logical-Log Files	14-5
Backing Up Logical-Log Files	14-6
Backing Up Blobspaces	14-6
Backing Up Sbspaces	14-7

Switching to the Next Logical-Log File	14-7
Freeing a Logical-Log File	14-8
Deleting a Log File with Status D	14-8
Freeing a Log File with Status U	14-9
Freeing a Log File with Status U-B or F	14-9
Freeing a Log File with Status U-C or U-C-L	14-10
Freeing a Log File with Status U-B-L	14-10
Monitoring Logging Activity	14-10
Monitoring the Logical Log for Fullness	14-11
Monitoring Temporary Logical Logs	14-13
Using SMI Tables	14-13
Using ON-Monitor	14-14
Monitoring Log-Backup Status	14-14
Allocating Log Files	14-14
Adding Logs Dynamically	14-14
Adding Logical-Log Files Manually	14-16
Dropping Logical-Log Files	14-18
Changing the Size of Logical-Log Files	14-20
Moving a Logical-Log File to Another Dbspace	14-20
Changing Logging Configuration Parameters	14-21
Using ON-Monitor to Change LOGFILES	14-22
Displaying Logical-Log Records	14-23
Monitoring Events for Dynamically Added Logs	14-23
Setting High-Watermarks for Rolling Back Long Transactions	14-25
Long-Transaction High-Watermark (LTXHWM)	14-26
Exclusive Access, Long-Transaction High-Watermark (LTXEHWM)	14-26
Adjusting the Size of Log Files to Prevent Long Transactions	14-27
Recovering from a Long Transaction Hang	14-27

Chapter 15 **Physical Logging, Checkpoints, and Fast Recovery**

In This Chapter	15-3
Critical Sections	15-3
Physical Logging	15-4
Fast Recovery Use of Physically-Logged Pages	15-4
Backup Use of Physically-Logged Pages	15-4
Database Server Activity That Is Physically Logged	15-5
Size and Location of the Physical Log	15-6
Specifying the Location of the Physical Log	15-6
Estimating the Size of the Physical Log.	15-7
Configuring the Size of the Physical Log	15-8
Details of Physical Logging	15-9
Page Is Read into the Shared-Memory Buffer Pool.	15-10
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	15-10
Change Is Reflected in the Data Buffer	15-10
Physical-Log Buffer Is Flushed to the Physical Log	15-10
Page Buffer Is Flushed	15-11
When a Checkpoint Occurs.	15-11
How the Physical Log Is Emptied	15-11
Checkpoints	15-11
Full Checkpoint.	15-12
Fuzzy Checkpoint	15-12
Events That Initiate a Fuzzy Checkpoint	15-14
Events That Initiate a Full Checkpoint	15-14
Sequence of Events in a Checkpoint.	15-15
Backup and Restore Considerations.	15-18
Fast Recovery	15-18
Need for Fast Recovery	15-18
Situations When Fast Recovery Is Initiated	15-19
Details of Fast Recovery After A Full Checkpoint	15-20
Details of Fast Recovery After A Fuzzy Checkpoint	15-23

Chapter 16	Managing the Physical Log	
	In This Chapter	16-3
	Changing the Physical-Log Location and Size	16-3
	Preparing to Make the Changes.	16-4
	Checking For Adequate Contiguous Space	16-4
	Using a Text Editor to Change Physical-Log Location or Size	16-5
	Using onparams to Change Physical-Log Location or Size	16-5
	Using ON-Monitor to Change Physical-Log Location or Size.	16-6
	Monitoring Physical and Logical-Logging Activity	16-6
	Sample onstat -l Output	16-8
	Monitoring Checkpoint Information	16-8
	Forcing a Full Checkpoint.	16-9
	Forcing a Fuzzy Checkpoint	16-10
	Using SMI Tables.	16-11

Section IV Fault Tolerance

Chapter 17	Mirroring	
	In This Chapter	17-3
	Mirroring	17-3
	Benefits of Mirroring	17-4
	Costs of Mirroring	17-4
	Consequences of Not Mirroring.	17-4
	Data to Mirror.	17-5
	Alternatives to Mirroring	17-6
	Mirroring Process	17-7
	Creation of a Mirrored Chunk	17-7
	Mirror Status Flags	17-8
	Recovery	17-9
	Actions During Processing	17-9
	Result of Stopping Mirroring.	17-11
	Structure of a Mirrored Chunk	17-11

Chapter 18

Using Mirroring

In This Chapter	18-3
Preparing to Mirror Data	18-3
Enabling the MIRROR Configuration Parameter	18-4
Changing the MIRROR Parameter with ON-Monitor.	18-4
Allocating Disk Space for Mirrored Data	18-5
Linking Chunks	18-5
Relinking a Chunk to a Device After a Disk Failure	18-5
Using Mirroring	18-6
Mirroring the Root Dbspace During Initialization	18-7
Changing the Mirror Status.	18-7
Managing Mirroring	18-8
Starting Mirroring for Unmirrored Storage Spaces.	18-8
Starting Mirroring for New Storage Spaces	18-9
Adding Mirrored Chunks	18-10
Taking Down a Mirrored Chunk	18-10
Recovering a Mirrored Chunk.	18-11
Ending Mirroring	18-11

Chapter 19

High-Availability Data Replication

In This Chapter	19-3
High-Availability Data Replication	19-4
Type of Data Replicated	19-4
Advantages of Data Replication	19-5
How HDR Works	19-9
How Data Initially Replicates	19-9
Reproduction of Updates to the Primary Database Server	19-10
Threads That Handle HDR	19-15
Checkpoints Between Database Servers	19-16
How Data Synchronization Is Tracked	19-16
HDR Failures	19-17
HDR Failures Defined	19-17
Detection of HDR Failures	19-18
Actions When an HDR Failure Is Detected	19-18
Considerations After HDR Failure	19-19
Redirection and Connectivity for Data-Replication Clients	19-21
Designing Clients for Redirection	19-21
Directing Clients Automatically with DBPATH	19-22
Directing Clients with the Connectivity Information	19-23
Directing Clients with INFORMIXSERVER	19-27

Handling Redirection Within an Application	19-28
Comparing Different Redirection Mechanisms	19-31
Designing HDR Clients	19-31
Setting Lock Mode to Wait for Access to Primary Database Server	19-32
Designing Clients to Use the Secondary Database Server	19-33

Chapter 20 Using High-Availability Data Replication

In This Chapter	20-3
Planning for HDR	20-3
Configuring a System for HDR	20-4
Meeting Hardware and Operating-System Requirements	20-4
Meeting Database and Data Requirements	20-5
Meeting Database Server Configuration Requirements	20-5
Configuring HDR Connectivity	20-9
Starting HDR for the First Time	20-9
Performing Basic Administration Tasks	20-14
Changing Database Server Configuration Parameters	20-14
Backing Up Storage Spaces and Logical-Log Files	20-15
Changing the Logging Mode of Databases	20-15
Adding and Dropping Chunks and Storage Spaces	20-16
Renaming Chunks	20-16
Saving Chunk Status on the Secondary Database Server	20-17
Using and Changing Mirroring of Chunks	20-18
Managing the Physical Log	20-19
Managing the Logical Log	20-19
Managing Virtual Processors	20-19
Managing Shared Memory	20-20
Changing the Database Server Mode	20-20
Changing the Database Server Type	20-21
Monitoring HDR Status	20-22
Restoring Data After Media Failure Occurs	20-24
Restoring After a Media Failure on the Primary Database Server	20-25
Restoring After a Media Failure on the Secondary Database Server	20-26
Restarting HDR After a Failure	20-27
Restarting After Critical Data Is Damaged	20-27
Restarting If Critical Data Is Not Damaged	20-30

Chapter 21 Consistency Checking

In This Chapter	21-3
Performing Periodic Consistency Checking	21-4
Verifying Consistency	21-4
Monitoring for Data Inconsistency	21-7
Retaining Consistent Level-0 Backups	21-9
Dealing with Corruption	21-9
Finding Symptoms of Corruption	21-9
Fixing Index Corruption.	21-10
Fixing I/O Errors on a Chunk	21-10
Collecting Diagnostic Information	21-11
Disabling I/O Errors	21-12
Monitoring the Database Server for Disabling I/O Errors	21-13
Using the Message Log to Monitor Disabling I/O Errors	21-13
Using Event Alarms to Monitor Disabling I/O Errors	21-14
Using No Bad-Sector Mapping	21-15

Section V Distributed Data

Chapter 22 Multiphase Commit Protocols

In This Chapter	22-3
Transaction Managers	22-3
Using the TP/XA Library With a Transaction Manager	22-4
Using Microsoft Transaction Server (MTS/XA)	22-4
Using Loosely-Coupled and Tightly-Coupled Modes.	22-5
Two-Phase Commit Protocol	22-5
When the Two-Phase Commit Protocol Is Used.	22-6
Two-Phase Commit Concepts	22-7
Phases of the Two-Phase Commit Protocol	22-8
How the Two-Phase Commit Protocol Handles Failures.	22-9
Presumed-Abort Optimization	22-10
Independent Actions	22-11
Situations That Initiate Independent Action	22-11
Possible Results of Independent Action	22-12
The Heuristic Rollback Scenario	22-15
The Heuristic End-Transaction Scenario	22-19
Monitoring a Global Transaction	22-22
Two-Phase Commit Protocol Errors	22-23

Two-Phase Commit and Logical-Log Records	22-23
Logical-Log Records When the Transaction Commits	22-25
Logical-Log Records Written During a Heuristic Rollback.	22-27
Logical-Log Records Written After a Heuristic End Transaction	22-29
Configuration Parameters Used in Two-Phase Commits	22-31
Function of the DEADLOCK_TIMEOUT Parameter.	22-31
Function of the TXTIMEOUT Parameter.	22-31
Heterogeneous Commit Protocol	22-32
Gateways That Can Participate in a Heterogeneous Commit Transaction	22-33
Enabling and Disabling of Heterogeneous Commit	22-34
How Heterogeneous Commit Works	22-34
Implications of a Failed Heterogeneous Commit	22-36

Chapter 23 Recovering Manually from Failed Two-Phase Commit

In This Chapter	23-3
Determining If Manual Recovery Is Required	23-3
Determining If a Transaction Was Implemented Inconsistently	23-4
Determining If the Distributed Database Contains Inconsistent Data	23-6
Deciding If Action Is Needed to Correct the Situation	23-9
Example of Manual Recovery	23-10

Appendix A Notices

Index

Introduction

In This Introduction	3
About This Manual	3
Types of Users	3
Software Dependencies	4
Assumptions About Your Locale	4
Demonstration Database	5
New Features in Dynamic Server, Version 9.4	6
Database Server Usability Enhancements	6
Security Enhancements	7
Features from Dynamic Server 9.3	7
Performance Enhancements	7
SQL Enhancements	8
Other Significant Changes in Version 9.3.	8
Features from Dynamic Server 9.21	9
Organizational Changes to This Manual Since Version 9.2	9
Documentation Conventions	10
Typographical Conventions	10
Icon Conventions	11
Comment Icons	11
Feature, Product, and Platform Icons	11
Compliance Icons	12
Sample-Code Conventions	12
Additional Documentation	13
Related Reading	15

Compliance with Industry Standards. 16

IBM Welcomes Your Comments 16

In This Introduction

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

About This Manual

This manual describes concepts and procedures for configuring, administering, and using IBM Informix Dynamic Server or IBM Informix Dynamic Server with J/Foundation.

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

This section discusses the organization of the manual, the intended audience, and the associated software products that you must have to use the database server.

Types of Users

This manual is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Performance engineers

- Programmers in the following categories
 - Application developers
 - DataBlade module developers
 - Authors of user-defined routines

This manual is written with the assumption that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to the *Getting Started Guide* for your database server for a list of supplementary titles.

Software Dependencies

This manual is written with the assumption that you are using IBM Informix Dynamic Server or IBM Informix Dynamic Server with J/Foundation, Version 9.4, as your database server.

Assumptions About Your Locale

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

The examples in this manual are written with the assumption that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for date, time, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

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

Demonstration Database

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

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

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

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

New Features in Dynamic Server, Version 9.4

The following table provides information about the new features for IBM Informix Dynamic Server, Version 9.4, which this manual covers. For a description of all new features, see the *Getting Started Guide*.

Database Server Usability Enhancements

Version 9.4 includes new features that make the database server easier to install, use, and manage.

New Features	Reference
The ability to clean buffers faster because of acceptance of decimals in the LRU configuration parameters.	“Number of Pages Added to the MLRU Queues” on page 7-38.
The ability to send email and pagermail notifications of alarm events and degree of severity.	“Event-Alarm Parameters” on page 2-15. “Appendix C: Event Alarms” in the Administrator’s Reference
The ability to use chunks and extents to a maximum size of 4 terabytes.	“Using Large Chunks” on page 1-9. “Creating Storage Spaces and Chunks” on page 1-25. “Supporting Large Chunks” on page 1-26. “Chunks” on page 9-6. “Sample Layout When Performance Is Highest Priority” on page 9-56. “Sample Disk Layouts” on page 9-55.
The ability to use a maximum of 32,766 allowable chunks.	“Specifying the Maximum Size of Chunks” on page 10-13.
The ability of HDR to replicate extended types	“Type of Data Replicated” on page 19-4.

Security Enhancements

Version 9.4 adds the encryption communication support module (ENCCSM), which enables you to encrypt data transmissions over the network. This option provides complete data encryption with the OpenSSL library, with many configurable options.

New Features	Reference
Encryption of data transmissions over the network	“Communication Support Modules (CSMs)” on page 3-20

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>).

Features from Dynamic Server 9.3

The following features were introduced in IBM Informix Dynamic Server Version 9.3.

Performance Enhancements

Version 9.3 includes many new features that help you monitor and improve the performance of your database.

New Features	Reference
The <code>onstat -g stm</code> option	“Monitoring Transactions” on page 12-11
Dynamic addition of logical logs	Chapter 13, “Logical Log” Chapter 14, “Managing Logical-Log Files”

SQL Enhancements

Version 9.4 includes several new SQL statements that ease migration from non-Informix databases to Dynamic Server, Version 9.4.

New Features	Reference
Configurable default lock modes	“Lock Table” on page 7-21

Other Significant Changes in Version 9.3

The following lists significant changes to the *Administrator’s Guide*.

Changes to the Manual	Reference
IBM Informix Server Administrator (ISA) has many new features.	ISA online help
Use the VPCLASS configuration parameter instead of the AFF_NPROCS, AFF_SPROC, NOAGE, NUMAIOVPS, and NUMCPUVPS configuration parameters.	Chapter 6, “Managing Virtual Processors”

Features from Dynamic Server 9.21

These features were introduced in IBM Informix Dynamic Server, Version 9.21.

Features	Reference
Nonlogging (RAW) tables	“Table Types for Dynamic Server” on page 9-38
onpladm utility	“Loading Data Into a Table” on page 10-61
SQL statement cache enhancements: <ul style="list-style-type: none"> ■ new configuration parameters ■ new onstat -g ssc options ■ new onmode -W options for changing SQL statement cache parameters 	“Setting SQL Statement Cache Parameters” on page 8-11

Organizational Changes to This Manual Since Version 9.2

The *Administrator's Guide* has been reorganized as follows:

- The information on configuring the database server and overview of administration tasks is [Chapter 1, “Installing and Configuring the Database Server.”](#)
- The information on multiple residency and high availability on Windows has been moved to the *Installation Guide*.
- The information on managing database server operating modes and initializing the database server is in [Chapter 4, “Initializing the Database Server.”](#)
- The information on the physical log and checkpoints is in [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”](#)

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> italics <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface boldface	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
<code>monospace</code> <code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of product- or platform-specific information.
→	This symbol indicates a menu item. For example, “Choose Tools→Options ” means choose the Options item from the Tools menu.






Tip: When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.




Comment Icons

Comment icons identify three types of information, as the following table describes. This information always appears in *italics*.

Icon	Label	Description
	<i>Warning:</i>	Identifies paragraphs that contain vital instructions, cautions, or critical information
	<i>Important:</i>	Identifies paragraphs that contain significant information about the feature or operation that is being described
	<i>Tip:</i>	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons


Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
	Identifies information that is specific to IBM Informix ESQL/C
	Identifies information that is specific to the UNIX operating system
	Identifies information that applies to all Windows environments

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the information that applies ends at the next heading at the same or higher level. A ♦ symbol indicates the end of information that appears in one or more paragraphs within a section.

Compliance Icons

Compliance icons indicate paragraphs that provide guidelines for complying with a standard.

Icon	Description
	Identifies information that is specific to an ANSI-compliant database

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the compliance information ends at the next heading at the same or higher level. A ♦ symbol indicates the end of compliance information that appears in one or more paragraphs within a section.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single IBM Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
      WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```



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

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

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

Additional Documentation

IBM Informix Dynamic Server documentation is provided in a variety of formats:

- **Online manuals.** The documentation CD in your media pack allows you to print the product documentation. You can obtain the same online manuals at the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>.
- **Online help.** This facility can provide context-sensitive help, an error message reference, language syntax, and more.
- **Documentation notes and release notes.** Documentation notes, which contain additions and corrections to the manuals, and release notes are located in the directory where the product is installed. Please examine these files because they contain vital information about application and performance issues. The following table describes these files.

UNIX

On UNIX platforms, the following online files appear in the \$INFORMIXDIR/release/en_us/0333 directory.

Online File	Purpose
ids_admin_docnotes_9.40.html	The documentation notes file for your version of this manual describes topics that are not covered in the manual or that were modified since publication.
ids_unix_release_notes_9.40.html	The release notes file describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
ids_machine_notes_9.40.txt	The machine notes file describes any special actions that you must take to configure and use IBM Informix products on your computer. Machine notes are named for the product described.



Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix→ Documentation Notes or Release Notes** from the task bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals with information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

Machine notes do not apply to Windows platforms. ♦

- **Error message files.** IBM Informix software products provide ASCII files that contain all of the error messages and their corrective actions. For a detailed description of these error messages, refer to *IBM Informix Error Messages* in the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>. To read the error messages on UNIX, you can use the **finderr** command to display the error messages online. ♦
- To read error messages and corrective actions on Windows, use the **Informix Error Messages** utility. To display this utility, choose **Start→Programs→Informix** from the task bar. ♦

UNIX

Windows

Related Reading

For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started Guide*.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

IBM Welcomes Your Comments

To help us with future versions of our manuals, let us know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Send electronic mail to us at the following address:

`docinf@us.ibm.com`

This address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact Customer Services.

The Database Server

- Chapter 1** **Installing and Configuring the Database Server**
- Chapter 2** **Configuration Parameters**
- Chapter 3** **Client/Server Communications**
- Chapter 4** **Initializing the Database Server**

Section I



Installing and Configuring the Database Server

In This Chapter	1-5
Planning for the Database Server	1-6
Considering Your Priorities	1-6
Considering Your Environment	1-7
Configuring the Operating System	1-7
Configuring Windows Memory	1-8
Modifying UNIX Kernel Parameters	1-8
Allocating Disk Space	1-8
Using Large Chunks	1-9
Creating Chunk Files on UNIX	1-9
Raw or Unbuffered Disk Access.	1-9
Cooked Files	1-10
Providing NTFS Partitions in Windows	1-10
Setting Permissions, Ownership, and Group.	1-10
Creating Standard Device Names	1-11
Setting Environment Variables	1-12
Setting GLS Environment Variables.	1-13
Setting Environment Variables on UNIX	1-14
Setting Environment Variables on Windows.	1-15
Configuring Connectivity	1-16
The sqlhosts File on UNIX	1-16
Network-Configuration Files.	1-17
Network-Security Files.	1-17
The sqlhosts Registry on Windows	1-17
Configuring Connectivity Using ISA	1-17

Configuring the Database Server	1-18
Preparing the ONCONFIG Configuration File	1-18
Creating an ONCONFIG File on UNIX	1-18
Creating an ONCONFIG File on Windows	1-19
Using Server Setup in ISA to Customize Your Configuration	1-20
Using IBM Informix Server Administrator to Update the ONCONFIG File	1-20
Using the Instance Manager to Create a New Database Server Instance.	1-21
Configuring Java Support	1-21
Starting and Administering the Database Server	1-22
Starting the Database Server and Initializing Disk Space.	1-22
Preparing for Automatic Startup	1-23
Preparing for Automatic Startup on Windows	1-23
Preparing the UNIX Startup and Shutdown Scripts	1-23
Preparing to Connect to Applications	1-25
Creating Storage Spaces and Chunks	1-25
Supporting Large Chunks	1-26
Setting Up Your Backup System and Storage.	1-26
Setting Up ontape.	1-27
Setting up Your Storage Manager and ON-Bar	1-27
Performing Routine Administrative Tasks	1-27
Changing Database Server Modes	1-28
Backing Up Data and Logical-Log Files	1-28
Monitoring Database Server Activity	1-28
Checking for Consistency	1-28
Performing Additional Administrative Tasks	1-29
Using Mirroring	1-29
Managing Database-Logging Status.	1-29
Managing the Logical Log	1-30
Managing the Physical Log.	1-30
Managing Shared Memory	1-31
Managing Virtual Processors	1-31
Managing Parallel Database Query	1-32
Using Data Replication	1-32
High-Availability Data Replication.	1-32
Enterprise Replication	1-32

Using Auditing	1-33
Using Distributed Queries	1-33
Using Global Transactions	1-33
Using a Transaction Manager	1-33
Monitoring Database Server Activity	1-34
Event Alarms	1-34
IBM Informix Server Administrator (ISA)	1-35
Message Log	1-35
Specifying the Destination for Message-Log Messages	1-35
Monitoring the Message Log	1-35
ON-Monitor	1-36
oncheck Utility	1-36
onperf Tool	1-36
onstat Utility	1-37
SMI Tables	1-37
System Console	1-37
UNIX Operating-System Tools	1-38
Windows Event Viewer	1-38
Windows Performance Monitor	1-39
Windows Utilities	1-40

In This Chapter

After you install IBM Informix Dynamic Server, Version 9.4, you must configure it. The install instructions are in the *IBM Informix Dynamic Server Installation Guide for UNIX and Linux* and the *IBM Informix Dynamic Server Installation Guide for Microsoft Windows*.

Configuration refers to setting specific parameters that customize the database server for your data-processing environment: quantity of data, number of tables, types of data, hardware, number of users, and security needs.

The following list identifies the basic configuration requirements:

- Planning for the database server
- Configuring the operating system
- Allocating disk space
- Setting environment variables
- Configuring connectivity information
- Preparing the ONCONFIG configuration file
- Using **Server Setup** to customize configuration
- Starting and administering the database server
- Monitoring database server activity

Planning for the Database Server

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

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

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

Considering Your Priorities

As you prepare the initial configuration and plan your backup strategy, keep in mind the characteristics of your database server:

- Do applications on other computers use this database server instance?
- What is the maximum number of users that you can expect?
- To what extent do you want to control the environment of the users?
- Are you limited by resources for space, CPU, or availability of operators?
- Do you need the database server to run unsupervised?
- Does the database server usually handle many short transactions or fewer long transactions?
- Which new database server features or related products do you plan to use?

Considering Your Environment

Before you start the initial configuration of your database server, collect as much of the following information as possible. You might need the assistance of your system administrator to obtain this information:

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

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

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

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

- Do you need to upgrade the hardware and operating system? Is your operating system 32-bit or 64-bit?
- Operating-system shared memory and other resources

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

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

UNIX

Configuring the Operating System

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

The 32-bit version of Dynamic Server runs on a 64-bit or 32-bit operating system. The 64-bit version of Dynamic Server must run on a 64-bit operating system. For more information, see [“Memory Use on 64-Bit Platforms” on page 7-50](#).

Windows

Configuring Windows Memory

On Windows, you must create NTFS partitions and configure memory. Also see [“Providing NTFS Partitions in Windows” on page 1-10](#).

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

UNIX

Modifying UNIX Kernel Parameters

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

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

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

For background information on the role of the UNIX kernel parameters, see [Chapter 8, “Managing Shared Memory.”](#)

Allocating Disk Space

Configure your disks to obtain optimum performance with data marts and data warehouses. Disk I/O is the longest portion of the response time for an SQL operation. The database server offers parallel access to multiple disks on a computer. Before you allocate the disk space, study the information about disk space in your operating-system administration guide and see [“Disk Allocation for Chunks” on page 9-7](#).

Using Large Chunks

The size of chunks for dbspaces is 4 terabytes for a 2-kilobyte page. Chunks can reside anywhere in a 64-bit address space.

To activate the creation of large chunks, you must use the **onmode** utility. The **onmode** utility uses a **-BC** flag to control the availability of large chunks, chunks greater than 2 gigabytes.

When the database server is first migrated to version 9.4, large chunks are not allowed. You must first run **onmode -BC 1** to allow large chunks to be created. However, the server continues to write small chunks to dbspaces and blobpages in version 9.3 and earlier previous formats. The server writes to that dbspace or blobpage in the 9.4 version format only when a large chunk is added to a dbspace or blobpage.

To complete the conversion to version 9.4 format, you must run **onmode -BC 2**. This will cause all server writes to be in the 9.4 version format.

If, during server initialization, the root chunk is a large chunk, the **onmode -BC 1** step is skipped and **-BC 2** mode is automatically initialized.

UNIX

Creating Chunk Files on UNIX

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

Raw or Unbuffered Disk Access

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

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

Important: *It is strongly recommended that you use raw disk devices on UNIX to achieve better performance.*



Windows

To allocate disks for the database server

1. Configure a raw disk device for each disk.
2. Create standard device names or filenames.
3. Set permissions, ownership, and group for each raw disk device.

Cooked Files

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

Providing NTFS Partitions in Windows

On Windows, install the database server on a New Technology File System (NTFS) or File Allocation Table (FAT) partition. However, you must store all dbspaces, blobspaces, and sbspaces in NTFS files or on a physical drive or logical disk partition. It is recommended that you use NTFS files to simplify disk administration. For more information about NTFS files, refer to your Windows documentation and [“Disk Access on Windows” on page 9-7](#).

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

```
convert /fs:ntfs
```

Setting Permissions, Ownership, and Group

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

UNIX

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

Windows

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

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

UNIX

Creating Standard Device Names

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

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

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

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



Setting Environment Variables

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

To set the required environment variables

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

Tip: *It is recommended that you set the environment variables in the appropriate startup file for your shell file or Windows.*

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

[Figure 1-1](#) shows the environment variables that you must set before you access the database server or perform most administrative tasks.

Figure 1-1
Required Environment Variables

Environment Variable	Description
CLASSPATH	If you are using J/Foundation, specifies the location of javaphome/krakatoa.jar file so that Java Development Kit (JDK) can compile the Java source files.
INFORMIXDIR	Specifies the directory where you installed your Informix database server.
INFORMIXSERVER	Specifies the name of the default database server. It has the value specified for the DBSERVERNAME or DBSERVERALIASES configuration parameter.
JVPHOME	If you are using J/Foundation, specifies the directory where you installed the IBM Informix JDBC Driver.

(1 of 2)

Environment Variable	Description
ONCONFIG	<p>Specifies the name of the active ONCONFIG file. All users who use database server utilities such as onstat must set the ONCONFIG environment variable. Users running client applications do not need to set the ONCONFIG environment variable.</p> <p>If the ONCONFIG environment variable is not present, the database server uses configuration values from the onconfig file:</p> <p>On UNIX: \$INFORMIXDIR/etc/onconfig</p> <p>On Windows: %INFORMIXDIR%\etc\onconfig.std</p>
PATH	<p>Specifies the location of executable files.</p> <p>On UNIX: \$INFORMIXDIR/bin</p> <p>On Windows: %INFORMIXDIR%\bin</p>
TERM	<p>Enables DB-Access to recognize and communicate with the terminal that you are using. This environment variable is not required for initialization, but it must be set before you can run an application.</p>
TERMCAP TERMINFO INFORMIXTERM	<p>Specifies whether DB-Access should use the information in the termcap file or the terminfo directory. If required for your system, you might need assistance from the UNIX system administrator to set these variables because they are highly system dependent.</p>

(2 of 2)

GLS

Setting GLS Environment Variables

The following environment variables let you work with the Global Language Support (GLS) feature. Set these environment variables if you want to use a locale other than the default GLS locale, U.S. English:

- CLIENT_LOCALE
- DB_LOCALE
- SERVER_LOCALE
- DBLANG

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

UNIX

Setting Environment Variables on UNIX

Set UNIX environment variables in one of the following ways:

- At the system prompt on the command line
When you set an environment variable at the system prompt, you must reassign it the next time you log in to the system.
- In an environment-configuration file such as **\$INFORMIXDIR/etc/informix.rc** or **.informix**
An environment-configuration file is a common or private file where you can set environment variables for each database server user. Use of an environment-configuration file reduces the number of environment variables that you must set at the command line or in a shell file.
- In your **.profile** or **.login** file
When you set an environment variable in your **.login**, **.cshrc**, or **.profile** file, it is assigned automatically every time you log in to the system. For information about these files, refer to your operating-system manuals.

To override environment variables that have been automatically set, use a private environment-variable file, **~/.informix**, or assign new values to environment variables individually. For information about the **.informix** and **informix.rc** files, see the *Administrator's Reference*.

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

Figure 1-2 shows a sample setup file that contains environment variables for the **miami** database server. **LD_LIBRARY_PATH** is set to the location of the database server and ESQL/C library files.

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

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

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

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

Figure 1-2
Sample Setup File

Windows

Setting Environment Variables on Windows

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

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

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

Use this batch file to configure command-prompt utilities.

For more information, see the *IBM Informix Guide to SQL: Reference* and [“Creating an ONCONFIG File on Windows” on page 1-19](#).

Configuring Connectivity

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

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

You do not need to specify all possible network connections in the **sqlhosts** file or registry before you initialize the database server. But to make a new connection available, you must take the database server offline and then bring it back to online mode once again.

For detailed information about configuring connectivity, see [Chapter 3, “Client/Server Communications.”](#)

UNIX

The sqlhosts File on UNIX

On UNIX, the **sqlhosts** file contains connectivity information. The default location of this file is **\$INFORMIXDIR/etc/sqlhosts**. If you store the information in another location, you must set the **INFORMIXSQLHOSTS** environment variable. For more information, see [“The sqlhosts File and the SQLHOSTS Registry Key” on page 3-29.](#)

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

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

Use a text editor or ISA to edit the **sqlhosts** file. For more information, see [“Configuring Connectivity Using ISA” on page 1-17.](#)

Network-Configuration Files

In addition to the **sqlhosts** files, TCP/IP connections require entries in the **/etc/hosts** and **/etc/services** files. For IPX/SPX connections, the names of the auxiliary files depend on the hardware vendor.

Network-Security Files

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

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

Windows

The sqlhosts Registry on Windows

On Windows, the HKEY_LOCAL_MACHINE registry contains the **sqlhosts** information. The database server installation procedure prepares the registry information. It is recommended that you not edit the HKEY_LOCAL_MACHINE registry.

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

Configuring Connectivity Using ISA

Use IBM Informix Server Administrator (ISA) to configure connectivity information for Informix database servers and database server groups for Enterprise Replication. ISA allows you to edit the **sqlhosts** file on UNIX or the **sqlhosts** registry on Windows. For more information, see the ISA onscreen instructions or online help.

In ISA, select **Configuration→SQLHOSTS**.

Configuring the Database Server

The configuration parameters are stored in the **ONCONFIG** file. The product installation script sets the defaults for most of the configuration parameters.

For information about the configuration parameters and how to monitor them, see [Chapter 2, “Configuration Parameters”](#) and the chapter on configuration parameters in the *Administrator's Reference*. For information on the order of files that the database server checks for configuration values, refer to [“Process Configuration File”](#) on page 4-6.

Preparing the ONCONFIG Configuration File

The **onconfig.std** template file in the **etc** subdirectory of **INFORMIXDIR** contains initial values for many of the configuration parameters. Copy this template and tailor it to your specific configuration.

The template files contain initial values for many of the configuration parameters. You can use a text editor or ISA to change the configuration parameters in your **ONCONFIG** file.



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

If you omit parameters in your copy of the **ONCONFIG** file, the database server uses values in the **onconfig.std** file for the missing parameters during initialization.

Creating an ONCONFIG File on UNIX

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

To prepare the ONCONFIG file using a text editor

1. Copy and rename the **\$INFORMIXDIR/etc/onconfig.std** file and store it in the **etc** subdirectory. (Skip this step if you are using ISA.)
2. Use a text editor or ISA to edit the **ONCONFIG** configuration file.

UNIX

Windows

3. Set the ONCONFIG environment variable to the name of your new ONCONFIG file.
4. Initialize the database server if it is a new instance.
Otherwise, shut down and restart the database server.

Creating an ONCONFIG File on Windows

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

To prepare the ONCONFIG file using the Instance Manager

1. Use the Instance Manager to create a new database server instance.
For details, see [“Using the Instance Manager to Create a New Database Server Instance” on page 1-21](#).
2. Use a text editor or ISA to edit the ONCONFIG configuration file.
3. Set your ONCONFIG environment variable to the name of your new ONCONFIG file.
4. Shut down and restart the database server for the configuration changes to take effect.
 - a. From the **Service** control application window, select the Dynamic Server service and click the **Stop** button.
 - b. Click the **Start** button.

To use dbservername.cmd to change the ONCONFIG environment variable

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

Using Server Setup in ISA to Customize Your Configuration

Use **Server Setup** in ISA to configure your production database server and optionally, J/Foundation and database storage. For more information, see the ISA online help and onscreen instructions.

To use Server Setup to customize your configuration

1. Start your web browser and open the URL for ISA:
`http://<hostname><domain_name>:port_number/`
If you did not install or start ISA, see your *Installation Guide* for directions.
2. Log in with the username and password that you provided during installation.
3. On the main page, select the database server name.
4. From the ISA server view page, click **Server Setup**.

Using IBM Informix Server Administrator to Update the ONCONFIG File

Use ISA to edit configuration parameters and set environment variables for all database servers. ISA automatically creates the **ONCONFIG** file (no need to copy **onconfig.std**).

To use IBM Informix Server Administrator to update the ONCONFIG file

1. In ISA, display the setup file and edit the environment variables.
2. Select the database server name.
3. In ISA, display the **ONCONFIG** file and modify specific configuration parameters.
ISA automatically updates your **ONCONFIG** file.
4. Shut down and restart the database server for the changes to take effect.

Windows

Using the Instance Manager to Create a New Database Server Instance

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

To use the Instance Manager

1. Select **Server Instance Manager** from the Dynamic Server menu.
2. Click **Create New** to create a new instance of the database server and follow the instructions in the wizard.
3. To improve database server performance, choose a scheduling priority for the **oninit** process in the Instance Manager. The default scheduling priority for Windows is **Normal**.
4. Click **Delete Server** to delete a database server instance.

Configuring Java Support

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

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

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

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

Starting and Administering the Database Server

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

- Preparing to connect to applications.
- Starting the database server and initialize disk space.
- Creating storage spaces.
- Setting up your backup and restore system.
- Performing administrative tasks.

Starting the Database Server and Initializing Disk Space

To bring the database server to online mode, enter **oninit**.

If starting a new database server, use the **oninit** command with the **-i** flag to initialize the disk space and bring the database server online. ♦

On Windows, the database server runs as a service. Use the **Service** control application to bring the database server to online mode. To initialize the database server, click the Dynamic Server service and enter **-iy** in the **Startup Parameters** box.

Another way to initialize the database server on Windows is to use the following command where *dbservername* is the database server name:

```
starts dbservername -iy
```

♦



Warning: When you initialize disk space, all of the existing data in the database server is destroyed. Initialize disk space only when you are starting a new database server.

For a description of the initialization types and associated commands, refer to [Chapter 4, “Initializing the Database Server.”](#)

UNIX

Windows

Preparing for Automatic Startup

Prepare the operating-system registry or scripts to automatically start and stop the database server.

Windows

Preparing for Automatic Startup on Windows

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

To start the database server automatically when Windows starts

1. From the **Service** control application window, select the Dynamic Server service and click the **Startup** button.
2. Select **Automatic** in the **Status Type** dialog box.
3. In the **Log On As** dialog box, select **This Account** and verify that `informix` is in the text box.

UNIX

Preparing the UNIX Startup and Shutdown Scripts

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

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

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

To prepare the UNIX startup script

1. Set the **INFORMIXDIR** environment variable to the full pathname of the directory in which the database server is installed.
2. Set the **PATH** environment variable to include the `$INFORMIXDIR/bin` directory.
3. Set the **ONCONFIG** environment variable to the desired configuration file.

4. Set the **INFORMIXSERVER** environment variable so that the **sysmaster** database can be updated (or created, if needed).
5. Execute **oninit**, which starts the database server and leaves it in online mode.

If you plan to initialize multiple versions of the database server (multiple residency), you must reset **ONCONFIG** and **INFORMIXSERVER** and re-execute **oninit** for each instance of the database server.

6. If you are using IBM Informix Storage Manager (ISM) for managing database server backups, you must start the ISM server on each node.
For information about how to start the ISM server, refer to your *Installation Guide*.

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

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

To prepare the UNIX shutdown script

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

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

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

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

Preparing to Connect to Applications

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

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

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

Creating Storage Spaces and Chunks

You are responsible for planning and implementing the storage configuration. The way you distribute the data on disks affects the performance of the database server. A *chunk* is the same as a logical volume, logical unit, or regular file that has been assigned to the database server. The maximum size of an individual chunk is 4 terabytes. The number of allowable chunks is 32,766. A logical *storage space* is composed of one or more chunks.



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

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

You must create an sbspace if you are using the following functions:

- J/Foundation (to hold Java JAR files)
- Enterprise Replication (to hold spooled row data)
- Smart large objects (BLOB and CLOB data types)
- Multirepresentational data types (such as DataBlades or HTML data types)

For a description of storage spaces and other physical units such as tblspaces and extents, refer to [Chapter 9, “Data Storage,”](#) For a discussion of the allocation and management of storage spaces, refer to [“Disk-Space Parameters” on page 2-4](#) and [Chapter 10, “Managing Disk Space.”](#)

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

Supporting Large Chunks

When the 9.4 server is initially converted, it is in a preparatory 9.4 version format. This means that large chunks, chunks greater than 2 gigabytes, are not yet enabled. To support large chunks and large offsets to the maximum size of 4 terabytes and more than 2047 chunks, you must first run **onmode -BC 1**.

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

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

Setting Up Your Backup System and Storage

To back up and restore your data, choose either the ON-Bar or **ontape** utility. For more information on setting up and using ON-Bar or **ontape**, see [“Backup and Restore Parameters” on page 2-7](#) and the *IBM Informix Backup and Restore Guide*.

Setting Up ontape

If you use **ontape** as your backup tool, you must set up storage devices (tape drives) before you can back up and restore data. The **ontape** utility does not require a storage manager.

Setting up Your Storage Manager and ON-Bar

If you use ON-Bar as your backup tool, you must set up a storage manager and storage devices before you can back up and restore data.

ON-Bar is packaged with IBM Informix Storage Manager (ISM). The *storage manager* is an application that manages the storage devices and media that contain backups. The storage manager handles all media labeling, mount requests, and storage volumes. ISM can back up data to as many as four storage devices at a time. ISM stores data on simple tape drives, optical disk devices, and file systems. However, you can purchase a third-party storage manager if you want to use more sophisticated storage devices, backups to more than four storage devices at a time, or backups over a network.

When you plan your storage-space and logical-log backup schedule, make sure that the storage devices and backup operators are available to perform backups. For information about configuring and managing storage devices and media, refer to the *IBM Informix Storage Manager Administrator's Guide* or to your third-party storage-manager documentation.

Performing Routine Administrative Tasks

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

Changing Database Server Modes

The database server administrator is responsible for starting up and shutting down the database server by changing the mode. [“Database Server Operating Modes” on page 4-12](#) explains how to change database server modes.

Backing Up Data and Logical-Log Files

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

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

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

Monitoring Database Server Activity

The Informix database server design lets you monitor every aspect of the database server. [“Monitoring Database Server Activity” on page 1-34](#) provides descriptions of the available information, instructions for obtaining information, and suggestions for its use.

Checking for Consistency

It is recommended that you perform occasional checks for data consistency. For a description of these tasks, refer to [Chapter 21, “Consistency Checking.”](#)

Performing Additional Administrative Tasks

This section covers various administrative tasks that you would perform on a production database server.

Using Mirroring

When you use disk mirroring, the database server writes data to two locations. Mirroring eliminates data loss due to storage device failures. If mirrored data becomes unavailable for any reason, the mirror of the data is available immediately and transparently to users. For information on mirroring, refer to [Chapter 17, “Mirroring.”](#) For instructions on tasks related to mirroring, see [Chapter 18, “Using Mirroring.”](#)



Important: *It is recommended that you mirror critical dbspaces that contain logical-log files, the physical log, and the root dbspace.*

Managing Database-Logging Status

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

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

- STANDARD
- TEMP
- RAW

For more information, refer to [“Temporary Tables” on page 9-42](#) and [Chapter 11, “Logging.”](#) For information on how to change logging options, refer to [Chapter 12, “Managing Database-Logging Mode.”](#)

Managing the Logical Log

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

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

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

For more information, refer to [“Logging Parameters” on page 2-6](#) and [Chapter 13, “Logical Log.”](#) For instructions on creating and modifying the logical-log configuration, see [Chapter 14, “Managing Logical-Log Files.”](#) For information on backing up the logical log, see the *IBM Informix Backup and Restore Guide*.

Managing the Physical Log

You can change the size and location of the physical log. For more information about the physical log, refer to [“Physical Log Parameters” on page 2-7](#), [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery,”](#) and [Chapter 16, “Managing the Physical Log.”](#)

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

Managing Shared Memory

Managing shared memory includes the following tasks:

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

For information on how the database server uses shared memory, refer to [Chapter 7, “Shared Memory.”](#) For additional information, refer to [“Shared-Memory Parameters” on page 2-9](#) and [Chapter 8, “Managing Shared Memory.”](#)

Managing Virtual Processors

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

For an explanation of virtual processors, refer to [Chapter 5, “Virtual Processors and Threads.”](#) For additional information, refer to [“Virtual Processor Parameters” on page 2-13](#) and [Chapter 6, “Managing Virtual Processors.”](#)

Managing Parallel Database Query

You can control the resources that the database uses to perform decision-support queries in parallel. You need to balance the requirements of decision-support queries against those of online transaction processing (OLTP) queries. The resources that you need to consider include shared memory, threads, temporary table space, and scan bandwidth. For information on parallel database query (PDQ) and how fragmentation affects the performance of PDQ, refer to your *Performance Guide*. For information on the configuration parameters, see [“Decision-Support Parameters” on page 2-12](#).

Using Data Replication

Data replication refers to the process of representing database objects at more than one distinct site. Informix database servers support two methods of data replication: *High-Availability Data Replication* (HDR) and *IBM Informix Enterprise Replication* (ER).



Tip: You cannot use HDR and Enterprise Replication on the system at the same time.

High-Availability Data Replication

HDR supports synchronous replication of an entire database to another database server, providing a hot standby in case of a catastrophic computer failure. For more information, see [“High-Availability Data-Replication Parameters” on page 2-15, Chapter 19, “High-Availability Data Replication,” and Chapter 20, “Using High-Availability Data Replication.”](#)

Enterprise Replication

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

Using Auditing

If you intend to use database server auditing, you need to specify where audit records are stored, how to handle error conditions, and so on. You also might want to change how users are audited if you suspect that they are abusing their access privileges. For information on tasks related to auditing, refer to [“Auditing Parameters” on page 2-17](#) and the *IBM Informix Trusted Facility Guide*.

Using Distributed Queries

The database server allows users to query and update more than one database across multiple database servers. This type of query is called a *distributed query*. The database servers can reside on a single host computer, on the same network, or on a gateway.

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

Using Global Transactions

A *global transaction* is a transaction that involves more than one database server. Informix database servers support two types of global transactions:

- TP/XA with a transaction manager
- Two-phase commit

Informix uses a *two-phase commit protocol* to ensure that distributed queries are uniformly committed or rolled back across multiple database servers. For more information, refer to [Chapter 22, “Multiphase Commit Protocols.”](#)

Using a Transaction Manager

Transaction managers manage terminals and data recovery. The TP/XA library enables communication between a third-party transaction manager and Informix databases in an X/Open environment. For more information, see the *TP/XA Programmer’s Manual* and [“Transaction Managers” on page 22-3](#).

Monitoring Database Server Activity

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

Source of Information	UNIX	Windows
Event alarms	✓	✓
IBM Informix Server Administrator (ISA)	✓	✓
Message log	✓	✓
ON-Monitor	✓	
oncheck utility	✓	✓
onperf utility	✓	
onstat utility	✓	✓
SMI tables	✓	✓
System console	✓	✓
Windows Event Viewer		✓
Windows Performance Monitor		✓

The following sections explain each of these sources.

Event Alarms

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

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

IBM Informix Server Administrator (ISA)

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

For more information on ISA, see the utilities chapter in the *Administrator's Reference* and the ISA online help.

Message Log

The database server *message log* is an operating-system file. The messages contained in the database server message log do not usually require immediate action. To report situations that require your immediate attention, the database server uses the event-alarm feature. See [“Event Alarms” on page 1-34](#). You can view the message log in ISA.

Specifying the Destination for Message-Log Messages

To specify the message-log pathname, set the MSGPATH configuration parameter. The changes to MSGPATH take effect after you shut down and restart the database server. For more information about MSGPATH, see the chapter on configuration parameters in the *Administrator's Reference*.

Monitoring the Message Log

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

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

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

UNIX

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

For more information, see the chapter on the messages in the *Administrator's Reference*, *IBM Informix Error Messages*, and [“Additional Documentation” on page 13](#) of the Introduction.

ON-Monitor

ON-Monitor provides a simple way to monitor many aspects of the database server. Most of the monitoring functions are available under the **Status** menu. For more information, see the section about ON-Monitor in the *Administrator's Reference*.

oncheck Utility

The **oncheck** utility displays information about the database disk configuration and usage, such as the number of pages used for a table, the contents of the reserved pages, and the number of extents in a table. For more information about **oncheck**, see the utilities chapter in the *Administrator's Reference*.

UNIX

onperf Tool

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

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

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

onstat Utility

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

SMI Tables

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

System Console

The database server sends messages that are useful to the database server administrator by way of the *system console*. To specify the destination pathname of console messages, set the CONSOLE configuration parameter. For more information about CONSOLE, see the chapter about configuration parameters in the *Administrator's Reference*.

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

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

Windows

UNIX

UNIX Operating-System Tools

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

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

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

Windows

Windows Event Viewer

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

To display database server messages on Windows

1. Choose **Administrative Tools→Event Viewer**.
2. Choose **Log→ Security**.
3. Double-click any event for a more detailed message.

Windows

Windows Performance Monitor

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

To display the Performance Monitor on Windows, choose **Administrative Tools→Performance Monitor**.

Windows Utilities

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

Windows Utility	Description and Usage
<code>ixpasswd.exe</code>	<p>Changes the logon password for all services which log on as user informix. You can change the password interactively or on the command line using the -y option. This utility saves having to manually change the password for each service whenever you change the informix password.</p> <p>If you are logged on locally and run ixpasswd, it changes the password for services that log on as the local informix user. If you are logged on <i>domain</i> and run ixpasswd, it changes the password for services that log on as <i>domain\informix</i>.</p> <p>Usage: <code>ixpasswd [-y new_password]</code></p>
<code>ixsu.exe</code>	<p>Launches a command-line window that runs as the specified user. The user is a local user unless you specify a domain name. If you do not specify a user name, the default user is informix. You no longer need to log off as the current user and log on as informix to perform DBA tasks that need to be run as informix.</p> <p>The ixsu utility requires Advanced User Rights:</p> <ul style="list-style-type: none">■ Act as part of the operating system■ Increase quotas■ Replace a process-level token <p>To configure Advanced User Rights on Windows NT, select User Manager→Policies→User Rights and check the Advanced User Rights option. If you change the Advanced User Rights for the current user, you need to log off and log back on for the new rights to take effect.</p> <p>Usage: <code>ixsu [[domain\]username]</code></p> <p>The ixsu utility is equivalent to the Windows 2000 runas command. To use runas to run a command shell as another user:</p> <p>Usage: <code>runas /user:username cmd</code></p>
<code>ntchname.exe</code>	<p>Changes the registry entries for Dynamic Server from the old hostname to the new hostname. Run ntchname after you change the hostname. This utility does not change user environment variables.</p> <p>After you execute ntchname, edit the %INFORMIXDIR%\%INFORMIX-SERVER%.cmd file and change the INFORMIXSQLHOSTS entry to the new hostname.</p> <p>Usage: <code>ntchname old_name new_name</code></p>

Configuration Parameters

In This Chapter	2-3
Database Server Identification Parameters	2-3
Disk-Space Parameters	2-4
Root Dbspace	2-4
Mirror of Root Dbspace	2-5
Other Space-Management Parameters	2-5
Logging Parameters	2-6
Logical Log	2-6
Physical Log Parameters	2-7
Backup and Restore Parameters	2-7
Message-Log Parameters	2-8
Shared-Memory Parameters	2-9
Shared-Memory Size Allocation	2-9
Shared-Memory Space Allocation	2-10
Shared-Memory Buffer Control	2-11
SQL Statement Cache Usage	2-11
Decision-Support Parameters	2-12
Database Server Process Parameters	2-13
Virtual Processor Parameters	2-13
Time Intervals	2-14
Restore Parameters	2-14
High-Availability Data-Replication Parameters	2-15
Event-Alarm Parameters	2-15

Dump Parameters	2-16
Specialized Parameters	2-16
Auditing Parameters	2-17
Optical Media Parameters	2-17
UNIX Parameters	2-17
Monitoring Configuration Information	2-18

In This Chapter

This chapter provides an overview of the ONCONFIG configuration parameters that the database server uses and describes different ways of monitoring configuration information. The chapter can help you decide which parameters are most crucial for your particular environment and which parameters you can defer until you are tuning the performance of your database server. For details on each parameter, see the chapter on configuration parameters in the *Administrator's Reference*.

Database Server Identification Parameters

Use the SERVERNUM and DBSERVERNAME parameters to provide unique identification for each instance of the database server.

Configuration Parameter	Description
DBSERVERALIASES	<p>Specifies an alternate name or names for an instance of the database server. The maximum number of aliases is 32.</p> <p>For information about using DBSERVERALIASES to create multiple listen endpoints to which clients can connect, see “Specifying Listen and Poll Threads for the Client/Server Connection” on page 5-33.</p>
DBSERVERNAME	<p>Specifies the unique name of an instance of the database server. Use the DBSERVERNAME for your INFORMIXSERVER environment variable and in the sqlhosts information.</p>
SERVERNUM	<p>Specifies a unique integer for the database server instance. The database server uses SERVERNUM to determine the shared-memory segment addresses.</p>



Warning: Do not change the DBSERVERNAME configuration parameter without reinitializing the database server.

Disk-Space Parameters

The disk-space parameters control how the database server manages storage space.

Root Dbspace

The first storage space that you allocate is called the *root database space*, or *root dbspace*. It stores all the basic information that describes your database server. Use the following parameters to describe the root dbspace.

Configuration Parameter	Description
ROOTNAME	Specifies the name of the root dbspace. You can choose any descriptive name for ROOTNAME, but it is usually called rootdbs . For more information, see “Root Dbspace” on page 9-18 .
ROOTOFFSET	Specifies an offset. For information about when to set ROOTOFFSET, refer to “Specifying an Offset” on page 10-6 .
ROOTPATH	Specifies the pathname of the storage allocated to the root dbspace. For information on how to choose and allocate the storage, see “Allocating Disk Space” on page 10-6 .
ROOTSIZE	Specifies the amount of space allocated to the root dbspace. For information on how to choose an appropriate size for the root dbspace, see “Size of the Root Dbspace” on page 9-49 .

Mirror of Root Dbspace

Mirroring allows fast recovery from a disk failure while the database server remains in online mode. When mirroring is active, the same data is stored on two disks simultaneously. If one disk fails, the data is still available on the other disk. Use the following parameters to describe mirroring of the root dbspace.

Configuration Parameter	Description
MIRROR	Defines whether mirroring is enabled or disabled. For more information, see Chapter 18, “Using Mirroring.”
MIRRORPATH	Specifies the full pathname of the chunk that mirrors the initial chunk of the root dbspace.
MIRROROFFSET	Specifies the offset into the device that serves as the mirror for the initial root dbspace chunk. For more information, see “Specifying an Offset” on page 10-6.

Other Space-Management Parameters

Use the following parameters to specify how the database server should manage particular types of disk space.

Configuration Parameter	Description
DBSPACETEMP	Specifies a list of dbspaces that the database server can use for the storage of temporary tables. For more information, see “Creating a Temporary Dbspace” on page 10-16.
FILLFACTOR	Specifies how much to fill index pages when indexes are created. For more information, see your <i>Performance Guide</i> .
ONDBSPACEDOWN	Defines how the database server treats a disabled dbspace that is not a critical dbspace.

(1 of 2)

Configuration Parameter	Description
SBSPACENAME	Specifies the name of the default sbspace. For more information, see “Control of Where Data Is Stored” on page 9-16 .
SBSPACETEMP	Specifies the name of the default temporary sbspace. For more information, see “Temporary Sbspaces” on page 9-32 .
SYSSBSPACENAME	Specifies the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types.

(2 of 2)

Logging Parameters

Use the logging parameters to control the logical and physical logs.

Logical Log

The logical log contains a record of changes made to a database server instance. The logical-log records are used to roll back transactions, recover from system failures, and so on. The following parameters describe logical logging.

Configuration Parameter	Description
DYNAMIC_LOGS	Determines whether the database server allocates new logical-log files automatically. For more information, see Chapter 13, “Logical Log.”
LOGBUFF	Determines the amount of shared memory reserved for the buffers that hold the logical-log records until they are flushed to disk. For information on how to tune the logical-log buffer, see “Logical-Log Buffer” on page 7-19 .
LOGFILES	Specifies the number of logical-log files used to store logical-log records until they are backed up on disk. For more information, see “Estimating the Size and Number of Log Files” on page 14-4 .

(1 of 2)

Configuration Parameter	Description
LOGSIZE	Specifies the size of each logical-log file.
LTXHWM	Specifies the percentage of the available logical-log space that, when filled, triggers the database server to check for a long transaction. For more information, see “Setting High-Watermarks for Rolling Back Long Transactions” on page 14-25 .
LTXEHWM	Specifies the point at which the long transaction being rolled back is given exclusive access to the logical log.

(2 of 2)

Physical Log Parameters

The physical log contains images of all pages (units or storage) changed since the last checkpoint. The physical log combines with the logical log to allow fast recovery from a system failure. Use the following parameters to describe the physical log.

Configuration Parameter	Description
PHYSBUFF	Determines the amount of shared memory reserved for the buffers that serve as temporary storage space for pages about to be modified.
PHYSDBS	Specifies the name of the dbspace in which the physical log resides.
PHYSFILE	Specifies the size of the physical log.

For more information, see [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”](#)

Backup and Restore Parameters

Use ON-Bar or **ontape** to create storage-space and logical-log backups of database server data. To verify storage-space backups, use ON-Bar. For more information on ON-Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

If you use the **ontape** utility, use the following parameters to describe your tape devices. To use a tape to its full physical capacity, set TAPESIZE and LTAPESIZE to 0 to read/write to the end of the medium.

Configuration Parameter	Description
TAPEDEV	Specifies tape devices.
TAPEBLK	Specifies the block size of the tape device.
TAPESIZE	Specifies the maximum amount of data that should be written to each tape.
LTAPEDEV	Specifies tape devices.
LTAPEBLK	Specifies the block size of the tape device.
LTAPESIZE	Specifies the maximum amount of data that should be written to each tape.

Message-Log Parameters

The message files provide information about how the database server is functioning.

Configuration Parameter	Description
CONSOLE (UNIX)	Specifies the pathname for console messages. For additional information, refer to “System Console” on page 1-37 .
MSGPATH	Specifies the pathname of the database server message-log file. For more information, refer to “Message Log” on page 1-35 .

Shared-Memory Parameters

The shared-memory parameters affect database server performance.

Shared-Memory Size Allocation

Use the following parameters to control how and where the database server allocates shared memory.

Configuration Parameter	Description
SHMADD	Specifies the increment of memory that is added when the database server requests more memory.
SHMBASE	Specifies the shared-memory base address and is computer dependent. Do not change its value.
SHMTOTAL	Specifies the maximum amount of memory that the database server is allowed to use.
SHMVIRTSIZE	Specifies the size of the first piece of memory that the database server attaches.

For more information on these parameters, see [Chapter 7, “Shared Memory.”](#)

For platform-specific information on these database server shared-memory configuration parameters, refer to your machine notes file on UNIX or your release notes file on Windows.

Shared-Memory Space Allocation

Use the following parameters to control how space is allocated in shared memory.

Configuration Parameter	Description
BUFFERS	Specifies the number of shared-memory buffers available to the database server. See “Shared-Memory Buffer Pool” on page 7-17 .
CKPTINTVL	Specifies the maximum time interval allowed to elapse before a checkpoint. For more information, see Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”
DD_HASHMAX	Specifies the maximum number of entries for each hash bucket in the data-dictionary cache. For more information about setting DD_HASHMAX, refer to your <i>Performance Guide</i> .
DD_HASHSIZE	Specifies the number of hash buckets in the data-dictionary cache. For more information about setting DD_HASHSIZE, refer to your <i>Performance Guide</i> .
DEF_TABLE_LOCKMODE	Sets the lock mode to page or row for new tables. For more information, see the <i>IBM Informix Guide to SQL: Tutorial</i> .
LOCKS	Specifies the initial number of locks available to database server user processes during transaction processing. For more information, see Chapter 8, “Managing Shared Memory” and your <i>Performance Guide</i> .
PC_POOLSIZE	Specifies the number of UDRs (both SPL routines and external routines) that can be stored in the UDR cache. In addition, this parameter specifies the size of other database server caches, such as the typename cache and the opclass cache. For more information about setting PC_POOLSIZE, refer to your <i>Performance Guide</i> .
PC_HASHSIZE	Specifies the number of hash buckets for the UDR cache and other caches that the database server uses. For more information about setting PC_HASHSIZE, refer to your <i>Performance Guide</i> .
RESIDENT	Specifies whether shared-memory residency is enforced. For more information, see Chapter 8, “Managing Shared Memory” and your <i>Performance Guide</i> .
STACKSIZE	Specifies the stack size for database server user threads. For a discussion of the use of stacks, refer to “Stacks” on page 7-29 .

Shared-Memory Buffer Control

Use the following parameters to control the shared-memory buffer pool.

Configuration Parameter	Description
LRUS, LRU_MAX_DIRTY, LRU_MIN_DIRTY	Describes the shared-memory pool of pages (memory spaces) that the database server uses. These parameters relate to LRU (least recently used) queues. See “LRU Queues” on page 7-34 .
CLEANERS	Controls the number of threads used to flush pages to disk and return the pages to the shared-memory pool. See “Flushing Data to Disk” on page 7-40 .
RA_PAGES and RA_THRESHOLD	Control the number of disk pages that the database server reads ahead during sequential scans. See “Configuring the Database Server to Read Ahead” on page 7-39 .

SQL Statement Cache Usage

Use the following parameters to configure the SQL statement cache. For more information, see [“Setting SQL Statement Cache Parameters” on page 8-11](#).

Configuration Parameter	Description
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL STATEMENT CACHE can hold a parsed and optimized SQL STATEMENT.
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.

Decision-Support Parameters

When you configure virtual shared memory on your system, you must decide what portion to reserve for decision-support queries. Decision-support queries use large amounts of the virtual portion of shared memory to perform joins and sort operations.

Use the following parameters to control how decision-support queries are processed and to control the amount of memory that the database server allocates to decision-support queries. For more information about tuning these configuration parameters, refer to your *Performance Guide*.

Configuration Parameter	Description
DATASKIP	Controls whether the database server skips an unavailable table fragment.
DS_MAX_QUERIES	Specifies the maximum number of queries that can run concurrently.
DS_MAX_SCANS	Limits the number of PDQ scan threads that the database server can execute concurrently.
DS_TOTAL_MEMORY	Specifies the amount of memory available for PDQ queries. Set the DS_TOTAL_MEMORY configuration parameter to any value not greater than the quantity (SHMVIRTSIZE - 10 megabytes).
MAX_PDQPRIORITY	Limits the amount of resources that a query can use.
OPTCOMPIND	Advises the optimizer on an appropriate join strategy for your applications.

Database Server Process Parameters

Configuration parameters for database server processes describe the type of processors on your computer and specify the behavior of virtual processes.

Virtual Processor Parameters

Use the following parameters to specify the type of processors in your environment and to allocate the virtual processors.

You need to set the following parameters to specific values, depending upon the number of processors on your platform:

- MULTIPROCESSOR
- SINGLE_CPU_VP
- VPCLASS

For guidelines on setting these parameters, refer to [“Setting Virtual-Processor Configuration Parameters” on page 6-3](#).

Configuration Parameter	Description
MULTIPROCESSOR	Specifies the appropriate type of locking.
NETTYPE	Provides tuning options for each communications protocol.
SINGLE_CPU_VP	Specifies that the database server is using only one processor and allow the database server to optimize for that situation.
VPCLASS	Specifies a class of virtual processors, the number of virtual processors that the database server should start, the maximum number allowed, the processor affinity, and the priority aging. It is recommended that you use the VPCLASS parameter instead of NOAGE, NUMCPUVPS, NUMAIOVPS, AFF_NPROCS, or AFF_SPROCS. For more information, see “Specifying VPCLASS” on page 6-5 .

Time Intervals

Use the following parameters to control the time intervals that the database server uses while processing transactions.

Configuration Parameter	Description
DEADLOCK_TIMEOUT	Specifies the amount of time that the database server waits for a shared-memory resource during a distributed transaction.
HETERO_COMMIT	Specifies whether the database server uses heterogeneous commit transactions.
TXTIMEOUT	Specifies how long a participant waits to receive a <i>commit</i> instruction during a two-phase commit.
USEOSTIME	Controls the granularity of time reported by the database server.

Restore Parameters

Use the following parameters to control the number of threads that the database server allocates to offline and online logical recovery. For more information, see your *Performance Guide*.

Configuration Parameter	Description
OFF_RECVRY_THREADS	Specifies the number of recovery threads used during a cold restore.
ON_RECVRY_THREADS	Specifies the number of recovery threads used during fast recovery and a warm restore.

High-Availability Data-Replication Parameters

Use the High-Availability Data-replication (HDR) parameters to control the behavior of a pair of HDR servers. For more information, see [Chapter 19, “High-Availability Data Replication.”](#)

Configuration Parameter	Description
DRINTERVAL	Specifies the maximum time interval in seconds between flushing of the data-replication buffer.
DRLOSTFOUND	Specifies the pathname to a file that contains transactions committed on the primary database server but not committed on the secondary database server when the primary database server experiences a failure.
DRTIMEOUT	Specifies how long in seconds a database server in a data-replication pair waits for a transfer acknowledgment from the other database server in the pair.

Event-Alarm Parameters

The database server can execute a program if a noteworthy event occurs. Noteworthy events include failure of database, table, index, chunk or dbspace taken offline, internal subsystem failure, initialization failure, and detection of long transaction. You can receive notification of a noteworthy event through email or pagermail. For more information, refer to your *Administrator's Reference*.

Use the following parameter to specify what actions to take when noteworthy events occur.

Configuration Parameter	Description
ALARMPROGRAM	Specifies the location of a file that is executed when an event alarm occurs. You can set the ALARMPROGRAM parameter to back up logs automatically as they fill. For more information, refer to the <i>IBM Informix Backup and Restore Guide</i> .

UNIX

Dump Parameters

Use the following parameters to control the types and location of core dumps that are performed if the database server fails. For more information, see [“Monitoring for Data Inconsistency” on page 21-7](#).

Configuration Parameter	Description
DUMPCNT	Specifies the number of assertion failures for which a single thread dumps shared memory.
DUMPCORE	Controls whether assertion failures cause a virtual processor to dump core memory.
DUMPDIR	Specifies a directory where the database server places dumps of shared memory, gcore files, or messages from a failed assertion.
DUMPGCORE	If your operating system supports the gcore utility, an assertion failure causes the database server to call gcore .
DUMPSHMEM	Specifies that shared memory should be dumped on an assertion failure.

Specialized Parameters

Some parameters appear in the configuration file only when you use specialized features of the database server.

UNIX

Auditing Parameters

Use the auditing parameters only when you use the database server auditing features. The *IBM Informix Trusted Facility Guide* describes these parameters.

Configuration Parameter	Description
ADTERR	Specifies how the database server behaves when it encounters an error while it writes an audit record.
ADTMODE	Controls whether the database server or the operating system manages auditing of user actions.
ADTPATH	Specifies the directory in which the database server can save audit files.
ADTSIZE	Specifies the maximum size of an audit file.

Optical Media Parameters

Use the following configuration parameters when you use the Optical Subsystem. For more information about these parameters, refer to the *IBM Informix Optical Subsystem Guide*.

Configuration Parameter	Description
OPCACHEMAX	Specifies the size of the memory cache.
STAGEBLOB	Specifies the name of the blob space for storing simple large objects that are destined for optical disk.

IBM Informix Storage Manager allows you to back up information to optical media, but it does not allow the database server to access directly the data that is stored on the disks.

UNIX

UNIX Parameters

Some UNIX platforms have additional configuration parameters. For a description of these specialized parameters and instructions on using them, see your machine notes.

Monitoring Configuration Information

One of the tasks of the database server administrator is to keep records of the configuration. [Figure 2-1](#) describes methods of obtaining configuration information.

Figure 2-1
Monitoring Configuration Information

Command	Description
onstat -c	<p>Displays a copy of the ONCONFIG file. For more information, see “Configuring the Database Server” on page 1-18.</p> <p>Changes to the ONCONFIG file take effect when you shut down and restart the database server, also called <i>reinitializing shared memory</i>. If you change a configuration parameter but do not shut down and restart the database server, the effective configuration differs from what the onstat -c option displays.</p> <p>The values of the configuration parameters are stored in the file indicated by the ONCONFIG environment variable or, if you have not set the ONCONFIG environment variable, in \$INFORMIXDIR/etc/onconfig on UNIX or %INFORMIXDIR%\etc\onconfig.std on Windows.</p>
oncheck -pr	<p>Lists the reserved page. The database server also stores current configuration information in the PAGE_CONFIG reserved page.</p> <p>If you change the configuration parameters from the command line and run oncheck -pr without shutting down and restarting the database server, the configuration values that oncheck displays do not match the current values in the reserved pages. The oncheck utility returns a warning message.</p>
ON-Monitor (UNIX)	Select Status→Configuration to create a copy of the current configuration and store it in the directory and file that you specify. If you specify only a filename, the database server stores the file in the current working directory. Changes to the configuration parameters take effect when you shut down and restart the database server.
ISA	Displays or updates the configuration parameters.

Figure 2-2 shows sample output from the **oncheck -pr** command.

```
...  
  
Validating Informix database server reserved pages - PAGE_CONFIG  
  ROOTNAME                rootdbs  
  ROOTPATH                /home/dyn_srv/root_chunk  
  ROOTOFFSET              0  
  ROOTSIZE                8000  
  MIRROR                  0  
  MIRRORPATH  
  MIRROROFFSET            0  
  PHYSDBS                rootdbs  
  PHYSFILE                1000  
  LOGFILES                5  
  LOGSIZE                 500  
  MSGPATH                /home/dyn_srv/online.log  
  CONSOLE                 /dev/ttyp5  
...
```

Figure 2-2
*PAGE_CONFIG
Reserved Page*

Client/Server Communications

In This Chapter	3-3
Client/Server Architecture	3-3
Network Protocol	3-4
Network Programming Interface.	3-5
Windows Network Domain	3-5
Database Server Connection	3-6
Multiplexed Connection.	3-7
Connections That the Database Server Supports.	3-8
Local Connections	3-10
Shared-Memory Connections	3-10
Stream-Pipe Connections	3-11
Named-Pipe Connections	3-11
Local-Loopback Connections	3-12
Communication Support Services.	3-12
Connectivity Files	3-13
Network-Configuration Files	3-13
TCP/IP Connectivity Files	3-13
Multiple TCP/IP Ports	3-16
IPX/SPX Connectivity Files	3-16
Network Security Files	3-17
The hosts.equiv File	3-17
The netrc Information	3-18

Communication Support Modules (CSMs)	3-20
CSM Configuration File.	3-21
Format of the CSM Configuration File for Password Encryption	3-21
Format of the CSM Configuration File for Network Data Encryption	3-23
The sqlhosts File and the SQLHOSTS Registry Key	3-29
The sqlhosts File	3-29
Tools for Updating SQLHOSTS Information	3-30
The SQLHOSTS Registry Key	3-31
The sqlhosts Information	3-32
Connectivity Information	3-34
Database Server Name	3-34
The Connection Type Field	3-34
Host Name Field	3-37
Service Name Field	3-38
Options Field	3-40
Group Information.	3-48
Database Server Group	3-48
Alternatives for TCP/IP Connections	3-48
IP Addresses for TCP/IP Connections	3-48
Wildcard Addressing for TCP/IP Connections.	3-49
Port Numbers for TCP/IP Connections	3-53
ONCONFIG Parameters for Connectivity	3-53
DBSERVERNAME Configuration Parameter.	3-54
DBSERVERALIASES Configuration Parameter	3-54
NETTYPE Configuration Parameter.	3-55
Environment Variables for Network Connections	3-56
Examples of Client/Server Configurations.	3-56
Using a Shared-Memory Connection	3-57
Using a Local-Loopback Connection	3-58
Using a Network Connection	3-59
Using Multiple Connection Types	3-60
Accessing Multiple Database Servers	3-62
Using IBM Informix MaxConnect	3-63

In This Chapter

This chapter explains the concepts and terms that you need to understand in order to configure client/server communications. The chapter consists of the following parts:

- Description of client/server architecture
- Database server connection types
- Communication services
- Connectivity files
- ONCONFIG connectivity parameters
- Connectivity environment variables
- Examples of client/server configurations

Client/Server Architecture

IBM Informix products conform to a software design model called *client/server*. The client/server model allows you to place an application or *client* on one computer and the database *server* on another computer, but they can also reside on the same computer. Client applications issue requests for services and data from the database server. The database server responds by providing the services and data that the client requested.

You use a *network protocol* together with a *network programming interface* to *connect* and transfer data between the client and the database server. The following sections define these terms in detail.

Network Protocol

A network protocol is a set of rules that govern how data is transferred between applications and, in this context, between a client and a database server. These rules specify, among other things, what format data takes when it is sent across the network. An example of a network protocol is TCP/IP.

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

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

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

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

To determine the supported protocols for your operating system, see [“Database Server Connection” on page 3-6](#).

To specify which protocol the database server uses, set the `nettype` field in the `sqlhosts` file on UNIX. On Windows, set the `PROTOCOL` field in the `SQLHOSTS` registry key. For more information, see [“The sqlhosts File and the SQLHOSTS Registry Key” on page 3-29](#).

Network Programming Interface

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

Both client and database server environments must be configured with the same protocol if client/server communication is to succeed. However, some network protocols can be accessed through more than one network programming interface. For example, TCP/IP can be accessed through either TLI or sockets, depending on which programming interface is available on the operating-system platform. Therefore, a client using TCP/IP through TLI on one computer can communicate with a database server using TCP/IP with sockets on another computer, or vice versa. For an example, see [“Using a Network Connection” on page 3-59](#).

Windows

Windows Network Domain

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

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

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



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

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

For more information on domains, consult your Windows operating system manuals.



Important: The Informix trusted client mechanism is unrelated to the trust relationship that you can establish between Windows domains. Therefore, even if a client connects from a trusted Windows domain, the user must have an account in the domain on which the database server is running. For more information on how the database server authenticates clients, see [“Communication Support Services”](#) on page 3-12 and [“Network Security Files”](#) on page 3-17.

Database Server Connection

A *connection* is a logical association between two applications; in this context, between a client application and a database server. A connection must be established between client and database server *before* data transfer can take place. In addition, the connection must be maintained for the duration of the transfer of data.



Tip: The Informix internal communications facility is called Association Services Facility (ASF). If you see an error message that refers to ASF, you have a problem with your connections.

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

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

For more information on the CONNECT and DATABASE statements, see the *IBM Informix Guide to SQL: Syntax*.

Multiplexed Connection

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

To configure the database server to support multiplexed connections, you must include in the ONCONFIG file a special NETTYPE parameter that has a value of SQLMUX, as in the following example:

```
NETTYPE SQLMUX
```

To configure the automatic use of multiplexed connections by clients, the entry in the **sqlhosts** file or registry that the client uses for the database server connection must specify the value of `m=1` in the **options** field, as in the following example:

```
menlo ontlitcp valley jfk1 m=1
```

You do not need to make any changes to the sqlhosts file or registry that the database server uses. The client program does not need to make any special SQL calls to enable connections multiplexing. Connection multiplexing is enabled automatically when the ONCONFIG file and the **sqlhosts** file or SQLHOSTS registry key are configured appropriately. For information on the NETTYPE configuration parameter, refer to the chapter on configuration parameters in the *Administrator's Reference*. For more information on the **sqlhosts** file or registry, refer to [“The sqlhosts File and the SQLHOSTS Registry Key” on page 3-29](#).

The following limitations apply to multiplexed connections:

- Multithreaded client connections are not supported.
- Shared-memory connections are not supported.

- Connections to subordinate database servers (for distributed queries or data replication, for example) are not multiplexed.
- The ESQL/C `sqlbreak()` function is not supported.
- You can activate database server support for multiplexed connections only when the database server starts.

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

Connections That the Database Server Supports

The database server supports the following types of connections to communicate between client applications and a database server.

Connection Type	Windows	UNIX	Local	Network
Sockets	✓	✓	✓	✓
TLI (TCP/IP)		✓	✓	✓
TLI (IPX/SPX)		✓	✓	✓
Shared memory		✓	✓	
Stream pipe		✓	✓	
Named pipe	✓		✓	

UNIX

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

```
Machine Specific Notes:
=====

1. The following interface/protocol combinations(s) are supported
for this platform:

    Berkeley sockets using TCP/IP
```



To set up a client connection

1. Specify connectivity configuration parameters in your ONCONFIG file.
2. Set up appropriate entries in the connectivity files on your platform.
3. Specify connectivity environment variables in your UNIX initialization scripts or the local and domainwide Windows registries.
4. Define a dbserver group for your database server in the **sqlhosts** file or registry.

The following sections describe database server connection types in more detail. For detailed information about implementing the connections described in the following sections, refer to the following topics:

- [“Connectivity Files” on page 3-13](#)
- [“The sqlhosts Information” on page 3-32](#)
- [“ONCONFIG Parameters for Connectivity” on page 3-53](#)
- [“Environment Variables for Network Connections” on page 3-56](#)

UNIX

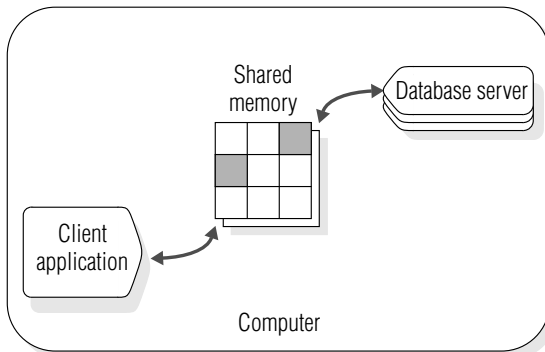
Local Connections

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

Shared-Memory Connections

A *shared-memory connection* uses an area of shared-memory as the *channel* through which the client and database server communicate with each other. [Figure 3-1](#) illustrates a shared-memory connection.

Figure 3-1
A Shared-Memory Connection



Shared memory provides fast access to a database server, but it poses some security risks. Errant or malicious applications could destroy or view message buffers of their own or of other local users. Shared-memory communication is also vulnerable to programming errors if the client application performs explicit memory addressing or overindexes data arrays. Such errors do not affect the database server if you use network communication or stream pipes. For an example of a shared-memory connection, refer to [“Using a Shared-Memory Connection”](#) on page 3-57.

A client cannot have more than one shared-memory connection to a database server.

UNIX

For information about the portion of shared memory that the database server uses for client/server communications, refer to [“Communications Portion of Shared Memory” on page 7-32](#). For additional information, you can also refer to [“How a Client Attaches to the Communications Portion” on page 7-11](#).

Stream-Pipe Connections

A *stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other. You can use stream-pipe connections any time that the client and the database server are on the same computer. For more information, refer to [“Network Protocol Entry” on page 3-36](#) and [“Shared-Memory and Stream-Pipe Communication” on page 3-38](#).

Stream-pipe connections have the following advantages:

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

Stream-pipe connections have the following disadvantages:

- Stream-pipe connections might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all platforms.

Windows

Named-Pipe Connections

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

Named pipes are supported for local connections to the database server.

Local-Loopback Connections

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

In a local-loopback connection, data appears to pass from the client application, out to the network, and then back in again to the database server. In fact, although the database server uses the network programming interface (TLI or sockets), the internal connection processes send the information directly between the client and the database server and do *not* put the information out on the network.

For an example of a local-loopback connection, see [“Using a Local-Loopback Connection” on page 3-58](#).

Communication Support Services

Communication support services include connectivity-related services such as the following security services:

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

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

The database server provides a default method of authentication, which is described in [“Network Security Files” on page 3-17](#). The database server uses the default authentication policy when you do not specify a communications support module.

The database server provides extra security-related communication support services through plug-in software modules called Communication Support Modules (CSM). For details, see [“Communication Support Modules \(CSMs\)” on page 3-20](#).

Connectivity Files

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

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

Network-Configuration Files

This section identifies and explains the use of network-configuration files on TCP/IP and IPX/SPX networks.

TCP/IP Connectivity Files

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

The network administrator maintains these files. When you add a host or a software service such as a database server, you need to inform the network administrator so that person can make sure the information in these files is accurate.

The **hosts** file needs a single entry for each network-controller card that connects a computer running an Informix client/server product on the network. Each line in the file contains the following information:

- Internet address (or ethernet card IP address)
- Host name
- Host aliases (optional)

Although the length of the host name is not limited in the **hosts** file, Informix limits the host name to 256 characters. [Figure 3-8 on page 3-49](#) includes a sample **hosts** file.

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

- Service name
IBM Informix products use this name to determine the port number and protocol for making client/server connections. The service name can have up to 128 characters.
- Port number and protocol
The port number is the computer port, and the protocol for TCP/IP is `tcp`.
The operating system imposes restrictions on the port number. User **informix** must use a port number equal to or greater than 1024. Only **root** users are allowed to use a port number less than 1024.
- Aliases (optional)

The service name and port number are arbitrary. However, they must be unique within the context of the file and must be identical on all computers that are running IBM Informix client/server products. The aliases field is optional. For example, a **services** file might include the following entry for a database server:

```
server2      1526/tcp
```

This entry makes `server2` known as the service name for TCP port 1526. A database server can then use this port to service connection requests. [Figure 3-6 on page 3-39](#) includes a sample **services** file.

For information about the **hosts** and **services** files, refer to your operating-system documentation.

UNIX*TCP/IP Connectivity Files on UNIX*

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

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

```
ypcat hosts
ypcat services
```

Windows*TCP/IP Connectivity Files on Windows*

You use information from the **hosts** and **services** network-configuration files to prepare the SQLHOSTS registry key for the TCP/IP network protocol. These files are in the following locations:

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

Alternately, you can configure TCP/IP to use the Domain Name Service (DNS) for host name resolutions. For information about these files, refer to your operating-system documentation.

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

Multiple TCP/IP Ports

To take advantage of multiple ethernet cards, take the following actions:

- Make an entry in the **services** file for each port the database server will use, as in the following example:

```
soc1      21/tcp
soc2      22/tcp
```

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

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

```
192.147.104.19      svc8
192.147.104.20      svc81
```

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

```
DBSERVERNAME  chicago1
DBSERVERALIASES  chicago2
```

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

```
chicago1 onsoctcp svc8 soc1
chicago2 onsoctcp svc81 soc2
```

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

UNIX

IPX/SPX Connectivity Files

To configure the database server to use the IPX/SPX protocol on a UNIX network, you must purchase IPX/SPX software and install it on the database server computer. Your choice of IPX/SPX software depends on the operating system that you are using. For some operating systems, the IPX/SPX software is bundled with software products based on NetWare for UNIX or Portable NetWare. In addition, for each of the UNIX vendors that distributes IPX/SPX software, you might find a different set of configuration files.

For advice on how to set configuration files for these software products, consult the manuals that accompany your IPX/SPX software.

Network Security Files

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

The hosts.equiv File

The **hosts.equiv** file lists the remote hosts and users that are trusted by the computer on which the database server resides. Trusted users, and users who log in from trusted hosts, can access the computer without supplying a password. The operating system uses the **hosts.equiv** file to determine whether a user should be allowed access to the computer without specifying a password. Informix requires a **hosts.equiv** file for its default authentication policy.

If a client application supplies an invalid account name and password, the database server rejects the connection even if the **hosts.equiv** file contains an entry for the client computer. You should use the **hosts.equiv** file only for client applications that do not supply a user account or password. On UNIX, the **hosts.equiv** file is in the **/etc** directory. On Windows, the **hosts.equiv** file is in the **\%WINDIR%\system32\drivers\etc** directory. If you do not have a **hosts.equiv** file, you must create one.

On some networks, the host name that a remote host uses to connect to a particular computer might not be the same as the host name that the computer uses to refer to itself. For example, the network host name might contain the full domain name, as the following example shows:

```
viking.informix.com
```

By contrast, the computer might refer to itself with the local host name, as the following example shows:

```
viking
```

If this situation occurs, make sure that you specify both host name formats in the **host.equiv** file.

Windows

To determine whether a client is trusted, execute the following statement on the client computer:

```
rlogin hostname
```

If you log in successfully without receiving a password prompt, the client is a trusted computer.

As an alternative, an individual user can list hosts from which he or she can connect as a trusted user in the **.rhosts** file. This file resides in the user's home directory on the computer on which the database server resides. On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the User Manager application. ♦

The netrc Information

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

On UNIX, the **netrc** information resides in the **.netrc** file in the user's home directory. Use any standard text editor to prepare the **.netrc** file. Windows maintains the **netrc** information in the registry keys. Use the Host Information tab of **setnet32** to edit the **netrc** information.

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

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

For information about the specific content of this file, refer to your operating-system documentation.

Windows

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

User Impersonation

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

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

File or Statement	Example
netrc information	machine trngpc3 login bruce password im4golf
CONNECT statement	CONNECT TO ol_trngpc3 USER bruce USING "im4golf"

Communication Support Modules (CSMs)

You can use communication support modules (CSMs) to enable two different kinds of encryption:

- The simple password CSM (SPWDCSM) provides password encryption.

This encryption protects a password when it must be sent between the client and the database server for authentication. SPWDCSM is available on all platforms.

- The encryption CSM (ENCCSM) enables you to encrypt data transmissions over the network.

This option provides complete data encryption with the openssl library, with many configurable options. A message authentication code (MAC) will be transmitted as part of the encrypted data transmission to ensure data integrity. A MAC is an encrypted message digest.

The encryption algorithms use openssl 0.9.6 as the code base.

Distributed queries can also be encrypted.

Enterprise Replication (ER) cannot use ENCCSM to provide encryption. To use network encryption with ER, set the encryption configuration parameters (for more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*).

Queries replicated with High-Availability Data Replication cannot be encrypted.

To use encryption, you must add a line to the CSS/CSM configuration file, **concsm.cfg**, and add an entry to the options column of the **sqlhosts** file or registry. The **concsm.cfg** file must contain an entry for each communications support module (of the same kind) that you are using.



Important: You cannot use both kinds of CSM simultaneously. For example, if you are using the SPWDCSM and decide to encrypt your network data, you must remove the entries for the SPWDCSM in your **concsm.cfg** and **sqlhosts** files.



Important: You cannot use either simple password CSM or encryption CSM over a multiplexed connection.

For information on specifying the CSM in the **sqlhosts** file or registry, see [“Communication Support Module Option” on page 3-43](#).

CSM Configuration File

The **concsbm.cfg** file is required if you use a communication support module (CSM). An entry in the file is a single line and is limited to 1024 characters. After you describe the CSM in the **concsbm.cfg** file, you can enable it in the options parameter of the **sqlhosts** file, as described in [“Communication Support Module Option” on page 3-43](#).

The **concsbm.cfg** file resides in the **\$INFORMIXDIR/etc** directory by default. If you want to store the file somewhere else, you can override the default location by setting the **INFORMIXCONCSBMCFG** environment variable to the full pathname of the new location. For information on setting the environment variable **INFORMIXCONCSBMCFG**, refer to the *IBM Informix Guide to SQL: Reference*.

Format of the CSM Configuration File for Password Encryption

The **concsbm.cfg** file entry has the following format:

```
cslname(lib-paths, "csm-global-option",
        "csm-connection-options")
```

The *cslname* variable is the name that you assign to the communications support module. The *lib-paths* parameter has the following format:

```
"client=lib-path-clientsdk-csm, server=lib-path-server-csm"
```

The *lib-path-clientsdk-csm* is the full pathname, including the filename, of the shared library that is the CSM of the client, and the client applications use this CSM to communicate with the database server. The CSM is normally installed in **\$INFORMIXDIR/lib/client/csm**.

The *lib-path-server-csm* is the full pathname, including the filename, of the shared library that is the CSM of the database server. The database server uses the CSM to communicate with the clients.

The database server CSM is usually installed in **\$INFORMIXDIR/lib/csm**. ♦

The database server CSM is usually installed in **\$INFORMIXDIR/bin**. ♦

UNIX

Windows

The following restrictions apply to the CSM pathnames:

- The following characters are not allowed to be part of the pathname:
 - = (equal sign)
 - " (double quote)
 - , (comma)
- White spaces cannot be used between = and the pathname or between pathname and , or " unless the white spaces are part of the pathname.

The *lib-paths* parameter can alternatively have the following format:

```
"lib-path-csm"
```

The *lib-path-csm* is the full pathname, including the filename, of the shared library that is the CSM. In this case, the same CSM is used by both the client applications and the database server.

The *csm-global* option is not used at this time for SPWDCSM.

The *csm-connection-options* option can contain the following options.

Setting	Result
p=1	The password is mandatory for authentication.
p=0	The password is not mandatory. If the client provides it, the password is encrypted and used for authentication.

An unknown option placed in *csm-connection-options* results in a context initialization error.

You can put a null value in the *csm-connection-options* field. For Client SDK before Version 2.3, if the *csm-connection-options* field is null, the default behavior is p=1. For Client SDK, Version 2.3 and later, if the *csm-connection-options* field is null, the default behavior is p=0.

SMI Tables and conccsm.cfg Setting

If you want to build the SMI tables when you bring up the database server (**oninit -i**), do not specify the **p=1** option in the database server CSM entry in the **conccsm.cfg** file. The **oninit** process does not have a password for the **informix** or **root** user ID. If you specify the **p=1** option in the **conccsm.cfg** file for the database server, you receive the following error message:

```
-5013 CSM: cannot obtain credential: authentication error.
```

To specify that the password is mandatory for the database server CSM when the SMI tables are not yet built

1. Do not specify the **p=1** option in the **conccsm.cfg** entry.
2. Bring up the database server with the **oninit -i** command to build the SMI tables.
3. Bring down the database server.
4. Specify the **p=1** option in the database server CSM entry in the **conccsm.cfg** file.
5. Start the database server again with the **oninit** command.

Example conccsm.cfg Entries for Password Encryption

The following two examples illustrate the two alternatives for parameters you must enter in the **conccsm.cfg** file to define the Simple Password Communication Support Module:

```
SPWDCSM("client=/usr/informix/lib/client/csm/libixspw.so,
        server=/usr/informix/lib/csm/libixspw.so", "", "")
```

```
SPWDCSM("/usr/informix/lib/csm/libixspw.so", "", "")
```

The following example shows the *csm-connection-options* field set to 0, so no password is necessary:

```
SPWDCSM("/work/informix/csm/libixspw.so", "", "p=0")
```

Format of the CSM Configuration File for Network Data Encryption

The basic format of the ENCCSM initialization string is:

```
ENCCSM("lib-paths", "cipher[options], mac[options], switch[options]")
```

UNIX

Windows

lib-paths Parameter

The *lib-paths* parameter provides the path and filename of the encryption DLL (**iecs09a.so**). It has the following format:

```
"client=lib-path-clientsdk-csm, server=lib-path-server-csm"
```

The *lib-path-clientsdk-csm* is the full pathname, including the filename, of the shared library that is the CSM of the client, and the client applications use this CSM to communicate with the database server. The CSM is normally installed in **\$INFORMIXDIR/lib/client/csm**.

The *lib-path-server-csm* is the full pathname, including the filename, of the shared library that is the CSM of the database server. The database server uses the CSM to communicate with the clients.

The database server CSM is usually installed in **\$INFORMIXDIR/lib/csm**. ♦

The database server CSM is usually installed in **\$INFORMIXDIR/bin**. ♦

The following restrictions apply to the CSM pathnames:

- The following characters are not allowed to be part of the pathname:
 - = (equal sign)
 - " (double quote)
 - , (comma)
- White spaces cannot be used between = and the pathname or between pathname and , or " unless the white spaces are part of the pathname.

The *lib-paths* parameter can alternatively have the following format:

```
"lib-path-csm"
```

The *lib-path-csm* is the full pathname, including the filename, of the shared library that is the CSM. In this case, the same CSM is used by both the client applications and the database server.

Major-tag Parameters

The other parameters (major tags) make up the string used to configure the ENCCSM:

- **cipher** — Defines all ciphers that can be used by the session.
- **mac** — Defines which MAC keys can be used by the session.
- **switch** — Defines the cipher and key change frequencies.

Each major tag has a parameter list that is enclosed by square brackets ([]). The general format of parameter list is *minor_tag:value*. If there can be multiple values for a minor tag, the list must be enclosed within angled brackets (<>).

The parameters are negotiated during the connect phase. If no common solution is obtained for the **mac** or **cipher** options, the connection attempt will fail.

The cipher Tag

The **cipher** tag can include one of the following minor tag options:

- **all**

Specifies to include all available ciphers and all available modes. There is no value list. For example:

```
cipher[all], ...
```

- **allbut:<list of ciphers to exclude>**

Specifies to include all ciphers except the ones in the list.

As there can be multiple entries, the list must be enclosed in angled brackets (<>). The list can include unique, abbreviated entries. For example, **bf** can represent bf-1, bf-2, and bf-3. However, if the abbreviation is the name of an actual cipher, then *only* that cipher is eliminated. Therefore, **des** eliminates only the DES cipher, but **de** eliminates des, des3, and desx.

For example:

```
cipher[allbut<ecb,des>], ...
```

```
cipher[allbut<cbc,bf>]
```



■ ***cipher:mode***

Specifies one or more cipher and mode. For example:

```
cipher [des3:cbc,des3:ofb]
```

Important: The encryption cipher and mode that will be used is randomly chosen among the ciphers that are common between the two servers. It is strongly recommended that you do not specify specific ciphers. For security reasons, all ciphers should be allowed. If a cipher is discovered to have a weakness, then that cipher can be eliminated by using the **allbut** option.

Ciphers supported at the time of going to press are shown below. For a full list, please check your Release Notes.

des	DES (64-bit key)	bf-1	Blow Fish (64-bit key)
des3	Triple DES	bf-2	Blow Fish (128-bit key)
desx	Extended DES (128-bit key)	bf-3	Blow Fish (192-bit key)



Important: The cipher *desx* can only be used in *cbc* mode.

Current supported modes are:

- **ecb** —Electronic Code Book
- **cbc** —Cipher Block Chaining
- **ocb** —Cipher Feedback
- **ofb** —Output Feedback

Because ECB mode is considered weak, it is only included if specifically requested. It is not included in the **all** or the **allbut** list.

The default value for the cipher field is:

```
cipher [allbut:<ecb>]
```

The mac Tag

The **mac** major tag defines the message authentication code (MAC) key files to be used during the MAC generation and the level of MAC generation utilized. The format for the **mac** tag is:

```
mac[levels:<list of MAC generation levels>,
    files:<full paths of MAC key files, builtin>
```

Use both of the following minor tags with the **mac** tag:

■ **levels**

Use the **levels** tag to specify a comma-separated list of MAC generation levels that the connection supports. The supported generation levels are:

- ❑ **high** —Uses SHA1 MAC generation on all messages.
- ❑ **medium** —Uses SHA1 MAC generation for all messages greater than 20 bytes long and XOR folding on smaller messages.
- ❑ **low** —Uses XOR folding on all messages.
- ❑ **off** —Does not use MAC generation.

The level is prioritized to the highest value. For example, if one node has a level of **high** and **medium** enabled and the other node has only **low** enabled, then the connection attempt will fail. The **off** entry should only be used between servers when it is guaranteed that there is a secure network connection.

■ **files**

Use the **files** parameter to specify a comma-separated list of the full path names of MAC key files. You can specify the built-in key by using the keyword **builtin**. Using the **builtin** option provides limited message verification (some validation of the received message and determination that it appears to have come from an IBM Informix Dynamic Server client or server), but the strongest verification is done by a site-generated MAC key file.

Each of the entries in the **mac** tag are prioritized and negotiated at connect time. The prioritization for the MAC key files is based on their creation time by the **GenMacKey** utility (the utility that is used to generate the MAC key). The **builtin** option has the lowest priority.

For example:

```
mac[levels:<high,low>,files:</usr/local/bin/mac1.dat,  
/usr/local/bin/mac2.dat,builtin>]
```

Because the MAC key files are negotiated, you can periodically change the keys using the following procedure:

1. Generate a new MAC key file by using the **GenMacKey** utility.
The utility, **GenMacKey**, is executed from the command line and has the syntax: **GenMacKey -o filename**.
2. Update the central server's configuration to include the location of the new MAC key file.
3. Distribute the new MAC key file.
4. Remove the old MAC key file entries from the configuration.

The default value for the **mac** tag is:

```
mac[builtin]
```



Tip: *If there are no MAC key files present, the built-in MAC key is used by default. However, by using a MAC key file, the default built-in MAC key is disabled.*

The switch Tag

The **switch** tag defines the frequency at which ciphers and or secret keys are re-negotiated. The longer that the secret key and encryption cipher remain in use, the more likely that the encryption rules might be broken by an attacker. To avoid this, cryptologists recommend periodically changing the secret key and cipher on long-term connections. The default time that this renegotiation occurs is once an hour. By using the **switch** tag, you can set the time in minutes when the renegotiation will occur.

The format of the **switch** tag is:

```
switch[cipher:value, key:value]
```

Use one or both of the following minor tags with the **switch** tag:

- **cipher**—Specifies the time in minutes between cipher renegotiation.
- **key**—Specifies the time in minutes between secret key renegotiation.

The time is specified in minutes, for example:

```
switch[cipher:120,key:20]
```

Example conccsm.cfg Entries for Network Data Encryption

The following two examples illustrate alternatives for parameters you enter in the **conccsm.cfg** file to define the encryption CSM. For example:

```
ENCCSM("$INFORMIXDIR/lib/csm/iencs09a.so", "cipher[allbut:<ecb,bf>]"
```

This configuration string states to use all available ciphers except for any of the BlowFish ciphers, and to not use any cipher in ECB mode.

Another example:

```
ENCCSM("$INFORMIXDIR/lib/csm/iencs09a.so",  
"cipher[des:cbc,des3:ofb,desx:cbc],switch[cipher:120,key:15]" )
```

This configuration string states to use the DES/CBC-mode, DES3/OFB-mode, and DESX/CBC-mode ciphers for this connection and also to switch the cipher being used every 120 minutes and renegotiate the secret key every 15 minutes.

The sqlhosts File and the SQLHOSTS Registry Key

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

For a detailed description of the **sqlhosts** information, refer to [“The sqlhosts Information” on page 3-32](#).

UNIX

The sqlhosts File

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

Each entry (each line) in the **sqlhosts** file contains the **sqlhosts** information for one database server. Use *white space* (spaces, tabs, or both) to separate the fields. Do not include any spaces or tabs *within* a field. To put comments in the **sqlhosts** file, start a line with the comment character (#). You can also leave lines completely blank for readability. Additional syntax rules for each of the fields are provided in the following sections, which describe the entries in the **sqlhosts** file. Use any standard text editor to enter information in the **sqlhosts** file.

Figure 3-2 shows a sample **sqlhosts** file.

Figure 3-2
Sample sqlhosts File

dbservername	nettype	hostname	servicename	options
menlo	onipcshm	valley	menlo	
newyork	ontlitcp	hill	dynsrvr2	s=2,b=5120
sales	ontlisp	knight	sales	k=0,r=0
payroll	onsoctcp	dewar	py1	
asia	group	—	—	e=asia.3
asia.1	ontlitcp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia

Tools for Updating SQLHOSTS Information

To manage the SQLHOSTS information, use one of the following tools:

- Text editor
- IBM Informix Server Administrator (ISA)
- **setnet32**



Tip: It is recommended that you use ISA to manage the SQLHOSTS connectivity information. Although **setnet32** allows you to set up database servers (nettype, hostname, servicename, and options), it does not allow you to set up database server groups.

Windows

The SQLHOSTS Registry Key

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

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

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

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

Location of the SQLHOSTS Registry Key

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

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

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

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

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

Figure 3-3 illustrates the location and contents of the SQLHOSTS registry key for the database server **payroll**.

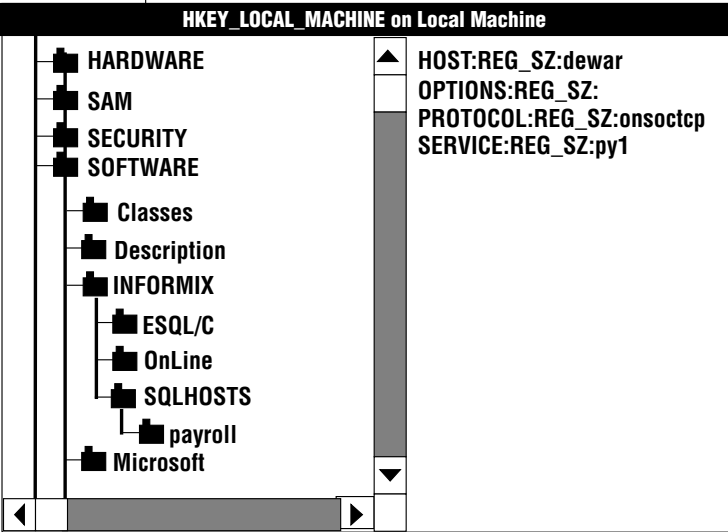


Figure 3-3
*sqlhosts Information in
the Windows Registry*

The sqlhosts Information

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

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

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

UNIX Field Name	Windows Field Name	Description of Connectivity Information	Description of Group Information
dbservername	Database server name key <i>or</i> database server group key	Database server name	Database server group name
nettype	PROTOCOL	Connection type	The word <i>group</i>
hostname	HOST	Host computer for the database server	<i>No information.</i> Use a hyphen as a placeholder in this field.
servicename	SERVICE	Alias for the port number	<i>No information.</i> Use a hyphen as a placeholder in this field.
options	OPTIONS	Options that describe or limit the connection	Group options

UNIX

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

Connectivity Information

The next section describes the connectivity information that is in each field of the `sqlhosts` file or SQLHOSTS registry key.

Database Server Name

The database server name field (**dbservername**) gives the name of the database server for which the connectivity information is being specified. Each database server across all of your associated networks must have a unique database server name. The **dbservername** field must match the name of a database server in the network, as specified by the DBSERVERNAME and DBSERVERALIASES configuration parameters in the ONCONFIG configuration file. For more information about these configuration parameters, refer to [“ONCONFIG Parameters for Connectivity” on page 3-53](#).

The **dbservername** field can include any printable character other than an uppercase character, a field delimiter, a newline character, or a comment character. It is limited to 128 characters.

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

The Connection Type Field

The connection-type field (**nettype** onUNIX or PROTOCOL on Windows) describes the type of connection that should be made between the database server and the client application or another database server. The field is a series of eight letters composed of three subfields, as [Figure 3-4](#) shows.

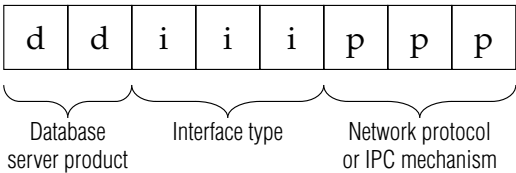


Figure 3-4
Format of the
Connection-Type
Field

The following sections describe the subfields of the connection-type field.

UNIX

Database Server Product

The first two letters of the connection-type field represent the database server product.

Product Subfield	Product
on or ol	The database server
dr (UNIX)	IBM Informix Enterprise Gateway with DRDA For information about DRDA, refer to the <i>IBM Informix Enterprise Gateway with DRDA User Manual</i> .

Interface Type

The middle three letters of the connection-type field represent the network programming interface that enables communications. For more information, see [“Network Programming Interface” on page 3-5](#).

Interface Subfield	Type of Interface
ipc	IPC (interprocess communications)
soc	Sockets
tli	TLI (transport layer interface)

Interprocess communications (IPC) are used only for communications between two processes running on the same computer.

Network Protocol Entry

The final three letters of the connection-type field represent the network protocol or specific IPC mechanism.

Protocol Subfield	Type of Protocol
imc	TCP/IP network protocol used with IBM Informix MaxConnect
nmp	Named-pipe communication
shm	Shared-memory communication
spx	IPX/SPX network protocol
str	Stream-pipe communication
tcp	TCP/IP network protocol

IPC connections use shared memory or stream pipes. The database server supports two network protocols: TCP/IP and IPX/SPX.

Figure 3-5 summarizes the possible connection-type values for database server connections. For more information on MaxConnect protocols, see [“Using IBM Informix MaxConnect” on page 3-63](#).

Figure 3-5
Summary of nettype Values

nettype Value (UNIX)	PROTOCOL Value (Windows)	Description	Connection Type
onipcshm		Shared-memory communication	IPC
onipcstr		Stream-pipe communication	IPC
	onipcnmp	Named-pipe communication	IPC
ontlitcp		TLI with TCP/IP protocol	Network
onsoctcp	onsoctcp	Sockets with TCP/IP protocol	Network
ontlisp		TLI with IPX/SPX protocol	Network

(1 of 2)

nettype Value (UNIX)	PROTOCOL Value (Windows)	Description	Connection Type
onsocimc		Sockets with TCP/IP protocol for communication with MaxConnect	Network
ontliimc		TLI with TCP/IP protocol for communication with MaxConnect	Network
SQLMUX	SQLMUX	Single network connection with multiple database connections.	Network

(2 of 2)

For information on the connection types for your platform, see [“Connections That the Database Server Supports” on page 3-8](#).

Host Name Field

The host name field (**hostname** on UNIX or **HOST** on Windows) contains the name of the computer where the database server resides. The name field can include any printable character other than a field delimiter, a newline character, or a comment character. The host name field is limited to 256 characters.

The following sections explain how client applications derive the values used in the host name field.

Network Communication with TCP/IP

When you use the TCP/IP network protocol, the host name field is a key to the **hosts** file, which provides the network address of the computer. The name that you use in the host name field must correspond to the *name* in the **hosts** file. In most cases, the host name in the **hosts** file is the same as the name of the computer. For information about the **hosts** file, refer to [“TCP/IP Connectivity Files” on page 3-13](#).

In some situations, you might want to use the actual Internet IP address in the host name field. For information about using the IP address, refer to [“IP Addresses for TCP/IP Connections” on page 3-48](#).

UNIX

Shared-Memory and Stream-Pipe Communication

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

UNIX

Network Communication with IPX/SPX

When you use the IPX/SPX network protocol, the **hostname** field must contain the name of the NetWare file server. The name of the NetWare file server is usually the UNIX *hostname* of the computer. However, this is not always the case. You might need to ask the NetWare administrator for the correct NetWare file server names.



Tip: NetWare installation and administration utilities might display the NetWare file server name in capital letters; for example, VALLEY. In the **sqlhosts** file, you can enter the name in either uppercase or lowercase letters.

Service Name Field

The interpretation of the service name field (**servicename** or SERVICE) depends on the type of connection that the connection-type field (**nettype** or PROTOCOL) specifies. The service name field can include any printable character other than a field delimiter, a newline character, or a comment character. The service name field is limited to 128 characters.

Network Communication with TCP/IP

When you use the TCP/IP connection protocol, the service name field must correspond to a service name entry in the **services** file. The port number in the **services** file tells the network software how to find the database server on the specified host. It does not matter what service name you choose, as long as you agree on a name with the network administrator.

Figure 3-6 shows the relationship between the **sqlhosts** file or registry and the **hosts** file, as well as the relationship of **sqlhosts** to the services file.

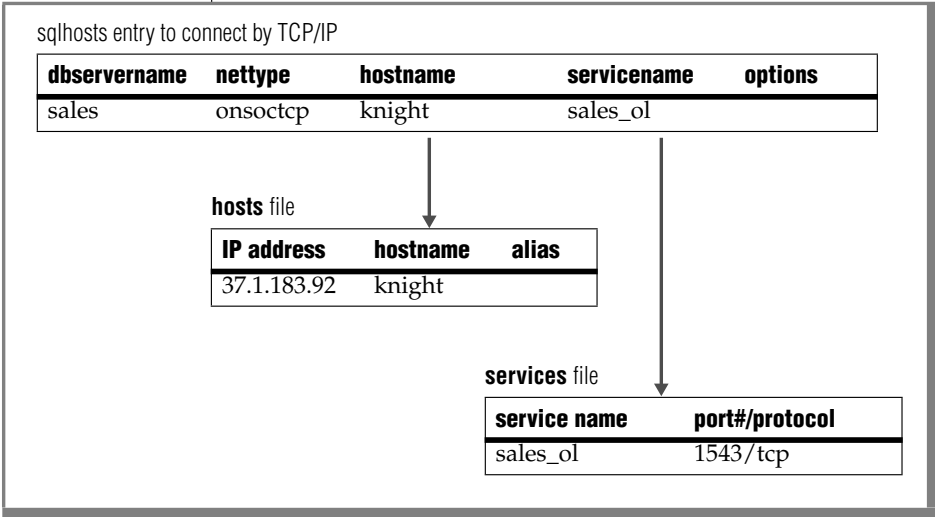


Figure 3-6
*Relationship of
sqlhosts File or
Registry to hosts
and services Files*

In some cases, you might use the actual TCP listen-port number in the service name field. For information about using the port number, refer to [“Port Numbers for TCP/IP Connections”](#) on page 3-53.

Windows

Named-Pipe Communication

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

UNIX

Shared-Memory and Stream-Pipe Communication

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

UNIX

Network Communication with IPX/SPX

A *service* on an IPX/SPX network is simply a program that is prepared to do work for you, such as the database server. For an IPX/SPX connection, the value in the **servicename** field can be an arbitrary string, but it must be unique among the names of services available on the IPX/SPX network. It is convenient to use the **dbservername** in the **servicename** field.

Options Field

The **options** field includes entries for the following features.

Option Name	Option Letter	Reference
Buffer size	b	page 3-41
Connection redirection	c	page 3-42
End of group	e	page 3-43
Group	g	page 3-44
Identifier	i	page 3-46
Keep-alive	k	page 3-46
Security	s (database server) r (client)	page 3-47
Communication support module	csm	page 3-43

When you change the values in the **options** field, those changes affect the next connection that a client application makes. You do not need to stop and restart the client application to allow the changes to take effect. However, a database server reads its own connectivity information *only* during initialization. If you change the options for the database server, you must reinitialize the database server to allow the changes to take effect.

Syntax Rules for the Options Field

Each item in the **options** field has the following format:

```
letter=value
```


You can combine several items in the **options** field, and you can include them in any order. The maximum length of the **options** field is 256 characters.

You can use either a comma or white space as the separator between options. You cannot use white space within an option.

The database server evaluates the **options** field as a series of columns. A comma or white space in the **options** field represents an end of a column. Client and database server applications check each column to determine whether the option is supported. If an option is not supported, you are not notified. It is merely ignored.

The following examples show both valid and invalid syntax.

Syntax	Valid	Comments
k=0,s=3,b=5120	Yes	Syntax is correct.
s=3,k=0 b=5120	Yes	Syntax is equivalent to the preceding entry. (White space is used instead of a comma.)
k=s=0	No	You cannot combine entries.

Buffer-Size Option

Use the buffer-size option (**b=value**) to specify in bytes the size of the communications buffer space. The buffer-size option applies only to connections that use the TCP/IP network protocol. Other types of connections ignore the buffer-size setting. You can use this option when the default size is not efficient for a particular application. The default buffer size for the database server using TCP/IP is 4096 bytes.

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



On many operating systems, the maximum buffer size supported for TCP/IP is 16 kilobytes. To determine the maximum allowable buffer size, refer to the documentation for your operating system or contact the technical-support services for the vendor of your platform.

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

Tip: *It is recommended that you use the default size for the communications buffer. If you choose to set the buffer size to a different value, set the client-side communications buffer and the database server-side communications buffer to the same size.*

Connection-Redirection Option

You can define a group of database servers, or aliases in the **sqlhosts** file on the client and have the client connections to the dbserver group try each of the group members with fail-over. The *connection-redirection* option (**c=value**) indicates the order in which the connection software chooses database servers, aliases, or coservers within a group.

Use the connection-redirection option in the following situations:

- To balance the load across multiple database server instances
- To use High-Availability Data Replication (HDR) to transfer over to a backup database server if a failure should occur

The following table shows the possible settings for the connection-redirection option.

Setting	Result
c=0	By default, a client application connects to the first database server instance listed in the dbserver group in the sqlhosts file. If the client cannot connect to the first instance, it attempts to connect to the second instance and so on.
c=1	The client application chooses a random starting point from which to connect to a list of dbserver group members.



Important: *The connection-redirection option is valid only in a dbserver group. For more information on the **g** option to specify dbserver groups, see [“Group Option” on page 3-44](#).*

Communication Support Module Option

Use the communication support module (CSM) option to describe the CSM for each database server that uses a CSM. If you do not specify the CSM option, the database server uses the default authentication policy for that database server. You can specify the same CSM option setting for every database server described in the **sqlhosts** file or registry, or you can specify a different CSM option or no CSM options for each **sqlhosts** entry.

The format of the CSM option is illustrated in the following example:

```
csn=(csnname,csn-connection-options)
```

The value of *csnname* must match a *csnname* entry in the **concsn.cfg** file. The *connection-options* parameter overrides the default *csn-connection* options specified in the **concsn.cfg** file. For information on the **concsn.cfg** file entry, refer to [“CSM Configuration File” on page 3-21](#).

The following example specifies that the ENCCSM communication support module will be used for the connection:

```
csn=(ENCCSM)
```

For more information on the CSM, refer to [“Database Server Connection” on page 3-6](#). For more information on the **concsn.cfg** file, refer to [“CSM Configuration File” on page 3-21](#).

End-of-Group Option

Use the end-of-group option (*e=dbservername*) to specify the ending database server name of a database server group. If you specify this option in an entry other than a database server group, it is ignored.

If no end-of-group option is specified for a group, the group members are assumed to be contiguous. The end of group is determined when an entry is reached that does not belong to the group, or at the end of file, whichever comes first. For an example of the end-of-group option, see [Figure 3-7 on page 3-45](#).

Group Option

When you define database server groups in the **sqlhosts** file or registry, you can use multiple related entries as one logical entity to establish or change client/server connections.

To create a database server group

1. Specify the name of the database server group to which the **sqlhosts** entry belongs (up to 18 characters) in the DBSERVERNAME field.
The database server group name can be the same as the initial DBSERVERNAME for the database server.
2. Place the keyword **group** in the connection type field.
3. The host name and service name fields are not used. Use dash (-) characters as null-field indicators for the unused fields. If you do not use options, you can omit the null-field indicators.

The options for a database server group entry are as follows:

- c = connection redirection
- e = end of group
- g = group option
- i = identifier option

To use database server groups for Enterprise Replication

Use the **i** and **g** options in Enterprise Replication. All database servers participating in replication must be a member of a database server group. Each database server in the enterprise must have a unique identifier, which is the server group. Make sure that the **sqlhosts** file is set up properly on each database server participating in replication.

For more information, see preparing the replication environment in the *IBM Informix Dynamic Server Enterprise Replication Guide*.

To use database server groups for High-Availability Data Replication

Use the **c**, **e**, and **g** options in High-Availability Data Replication (HDR). HDR requires two identical systems. For more information, see [“Directing Clients with the Connectivity Information” on page 19-23](#).



Important: Database server groups cannot be nested inside other database server groups, and database server group members cannot belong to more than one group.

The example in shows the following two groups: **asia** and **peru**. Group **asia** includes the following members:

- **asia.1**
- **asia.2**
- **asia.3**

Because group **asia** uses the end-of-group option (`e=asia.3`), the database server searches for group members until it reaches **asia.3**, so the group includes **usa.2**. Because group **peru** does not use the end-of-group option, the database server continues to include all members until it reaches the end of file.

Figure 3-7 shows examples of database server groups in the **sqlhosts** file.

Figure 3-7
Database Server Groups in sqlhosts File or Registry

dbservername	nettype	hostname	servicename	options
asia	group	–	–	e=asia.3
asia.1	ontlitzp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia
usa.2	ontlisp	node9	sv2	
asia.3	onsoctcp	node1	svc9	g=asia
peru	group	–	–	
peru.1	ontlitzp	node4	svc4	
peru.2	ontlitzp	node5	svc5	g=peru
peru.3	ontlitzp	node7	svc6	
usa.1	onsoctcp	37.1.183.92	sales_ol	k=1, s=0

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

■ INFORMIXSERVER

The value of **INFORMIXSERVER** for a client application can be the name of a database server group. However, you cannot use a database server group name as the value of **INFORMIXSERVER** for a database server or database server utility.

■ DBPATH

DBPATH can contain the names of database server groups as dbservernames.

Identifier Option

The identifier option (*i=number*) assigns an identifying number to a database server group. The identifier must be a positive numeric integer and must be unique within your network environment.

For more information on the use of the identifier option, refer to the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Keep-Alive Option

The keep-alive option is a network option that TCP/IP uses. It does not affect other types of connections. If you do not include the keep-alive option in the **options** field, the keep-alive feature is enabled by default. To set the keep-alive option on the database server side only, the client side only, or on both sides, specify **k=1** in the fifth column of the **sqlhosts** file. For most cases, it is recommended that you enable the keep-alive option.

The letter *k* identifies keep-alive entries in the **options** field, as follows:

k=0	disable the keep-alive feature
k=1	enable the keep-alive feature

When a connected client and server are not exchanging data and the keep-alive option is enabled, the network service checks the connection periodically. If the receiving end of the connection does not respond within the time specified by the parameters of your operating system, the network service immediately detects the broken connection and frees resources.

When the keep-alive option is disabled, the network service does not check periodically whether the connection is still active. If the opposite end of the connection terminates unexpectedly without any notification, as when a PC reboots, for example, the network service might never detect that the connection is broken.

Security Options

The security options let you control operating-system security-file lookups. The letter *s* identifies database server-side settings, and the letter *r* identifies client-side settings. You can set both options in the **options** field. A client ignores *s* settings, and the database server ignores *r* settings.

The following table shows the possible security option settings.

Setting	Result
r=0	Disables netrc lookup from the client side (no password can be supplied).
r=1	Enables netrc lookup from the client side (default setting for the client side).
s=0	Disables both hosts.equiv and rhosts lookup from the database server side (only incoming connections with passwords are accepted).
s=1	Enables only the hosts.equiv lookup from the database server side.
s=2	Enables only the rhosts lookup from the database server side.
s=3	Enables both hosts.equiv and rhosts lookup on the database server side (default setting for the database server side).

The security options let you control the way that a client (user) gains access to a database server. By default, an Informix database server uses the following information on the client computer to determine whether the client host computer is trusted:

- **hosts.equiv**
- **rhosts** information

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



For example, if you want to prevent end users from specifying trusted hosts in **rhosts**, you can set `s=1` in the **options** field of the **sqlhosts** file or SQLHOSTS registry key for the database server to disable the **rhosts** lookup.

Important: Do not disable the *hosts.equiv* lookup in database servers that are used in distributed database operations. That is, if you expect to use the database server in distributed processing, do not set `s=0` or `s=2`.

Group Information

The following section describes the fields of the **sqlhosts** file or registry for groups.

Database Server Group

A database server group allows you to treat multiple related database server entries as one logical entity to establish or change client/server connections. You can also use dbserver groups to simplify the redirection of connections to database servers. For more information on database server groups, see [“Group Option” on page 3-44](#).

Alternatives for TCP/IP Connections

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

IP Addresses for TCP/IP Connections

For TCP/IP connections (both TLI and sockets), you can use the actual IP address in the **hostname** field instead of the host name or alias found in the **hosts** file. The IP address is composed of four integers between 0 and 255, separated by periods. [Figure 3-8 on page 3-49](#) shows sample IP addresses and hosts from a **hosts** file.

Figure 3-8
A Sample hosts File

IP Address	Host Name	Host Alias(es)
555.12.12.12	smoke	
98.765.43.21	odyssey	
12.34.56.789	knight	sales

Using the IP address for knight from [Figure 3-8](#), the following two **sqlhosts** entries are equivalent:

```
sales    ontllitcp    12.34.56.789    sales_ol
sales    ontllitcp    knight          sales_ol
```

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

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

You can configure Windows to use either of the following mechanisms to resolve Internet Domain Addresses (*mymachine.informix.com*) to Internet Protocol addresses (149.8.73.14):

- Windows Internet Name Service
- Domain Name Server ♦

Wildcard Addressing for TCP/IP Connections

You can use wildcard addressing in the host name field when *both* of the following conditions are met:

- You are using TCP/IP connections.
- The computer where the database server resides has multiple network-interface cards (for example, three ethernet cards).

UNIX

Windows

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

Each IP address is associated with a unique host name. When a computer has multiple network-interface cards (NICs), as in [Figure 3-9 on page 3-51](#), the **hosts** file must have an entry for each interface card. For example, the **hosts** file for the **texas** computer might include these entries.

NIC	Internet IP Address	Host Name
Card 1	123.45.67.81	texas1
Card 2	123.45.67.82	texas2

You can use the wildcard (*) alone or as a prefix for a host name or IP address, as shown in [Figure 3-10 on page 3-52](#).

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

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

[Figure 3-9](#) shows a database server on a computer (**texas**) that has two network-interface cards. The two client sites use different network cards to communicate with the database server.

Figure 3-9
Using Multiple Network-Interface Cards

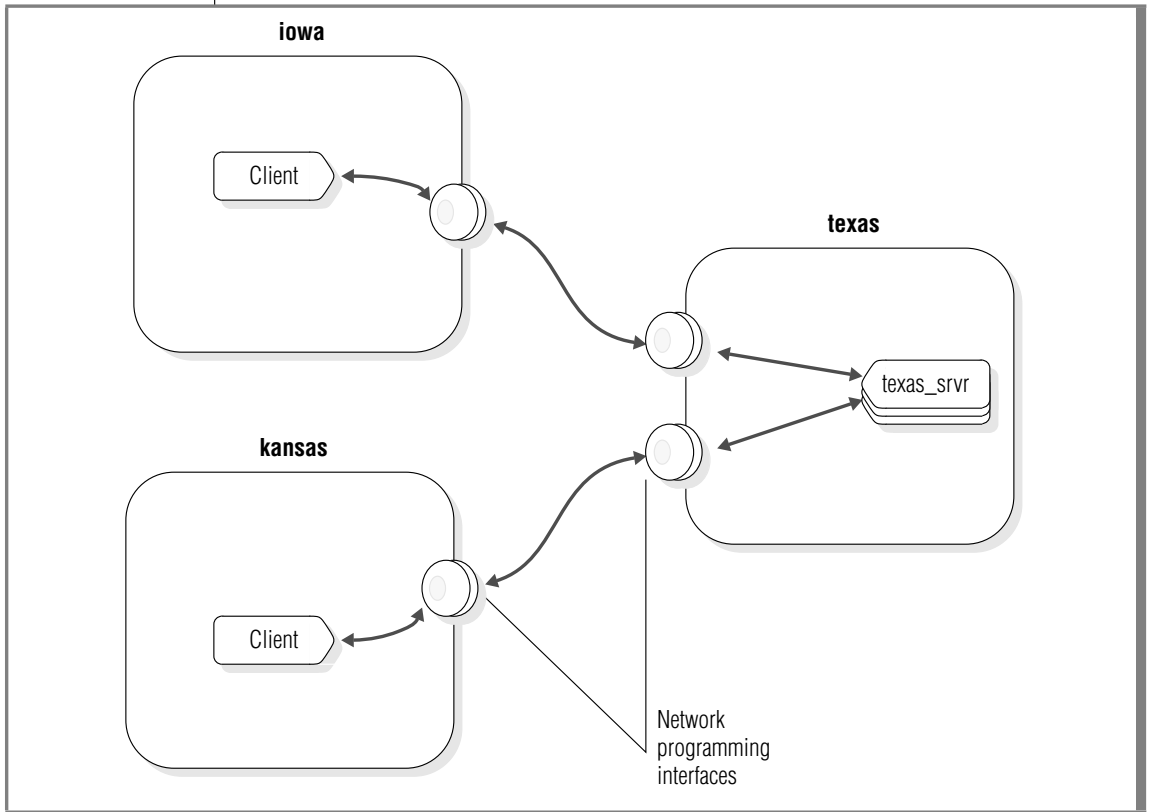


Figure 3-10 shows the connectivity information for the **texas_svr** database server.

Figure 3-10
Possible Connectivity Entries for the texas_svr Database Server

database server name	connection type	host name	service name
texas_svr	ontlitcp	*texas1	pd1_on
texas_svr	ontlitcp	*123.45.67.81	pd1_on
texas_svr	ontlitcp	*texas2	pd1_on
texas_svr	ontlitcp	*123.45.67.82	pd1_on
texas_svr	ontlitcp	*	pd1_on



Important: You can include only one of these entries.

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



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

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

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

Port Numbers for TCP/IP Connections

For the TCP/IP network protocol, you can use the actual TCP listen port number in the service name field. The TCP port number is in the **port#** field of the **services** file.

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

servername	nettype	hostname	servicename
sales	ontlircp	knight	1543

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

ONCONFIG Parameters for Connectivity

When you initialize the database server, the initialization procedure uses parameter values from the ONCONFIG configuration file. The following ONCONFIG parameters are related to connectivity:

- DBSERVERNAME
- DBSERVERALIASES
- NETTYPE

The next sections explain these configuration parameters.

DBSERVERNAME Configuration Parameter

The DBSERVERNAME configuration parameter specifies a name, called the *dbservername*, for the database server. For example, to assign the value *nyc_research* to *dbservername*, use the following line in the ONCONFIG configuration file:

```
DBSERVERNAME nyc_research
```

When a client application connects to a database server, it must specify a *dbservername*. The **sqlhosts** information that is associated with the specified *dbservername* describes the type of connection that should be made.

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

- In the **INFORMIXSERVER** environment variable
- In SQL statements such as **CONNECT**, **DATABASE**, **CREATE TABLE**, and **ALTER TABLE**, which let you specify a database environment
- In the **DBPATH** environment variable

Windows

In Windows, the DBSERVERNAME cannot simply be changed in the configuration file, because the registry stores information about the database server instance under the DBSERVERNAME. ♦

DBSERVERALIASES Configuration Parameter

The DBSERVERALIASES parameter lets you assign additional *dbservernames* to the same database server. The maximum number of aliases is 32.

[Figure 3-11](#) shows entries in an ONCONFIG configuration file that assign three *dbservernames* to the same database server instance.

Figure 3-11

Example of DBSERVERNAME and DBSERVERALIASES Parameters

DBSERVERNAME	sockets_srvr
DBSERVERALIASES	ipx_srvr, shm_srvr

The **sqlhosts** entries associated with the dbservernames from [Figure 3-11](#) could include those shown in [Figure 3-12](#). Because each dbservername has a corresponding entry in the **sqlhosts** file or SQLHOSTS registry key, you can associate multiple connection types with one database server.

Figure 3-12

Three Entries in the sqlhosts File for One Database Server in UNIX format

shm_srvr	onipcshm	my_host	my_shm
sockets_srvr	onsoctcp	my_host	port1
ipx_srvr	ontlisp	nw_file_server	ipx_srvr

Using the **sqlhosts** file shown in [Figure 3-12](#), a client application uses the following statement to connect to the database server using shared-memory communication:

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

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

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

NETTYPE Configuration Parameter

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

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

For more information about NETTYPE, refer to [“Network Virtual Processors” on page 5-31](#). For additional information about the NETTYPE configuration parameter, refer to the *Administrator’s Reference*.

Environment Variables for Network Connections

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

If the client application explicitly attaches to shared-memory segments, you might need to set **INFORMIXSHMBASE** (shared-memory base). For more information, refer to [“How a Client Attaches to the Communications Portion” on page 7-11](#).

The **INFORMIXSERVER** environment variable allows you to specify a default dbservername to which your clients will connect.

For more information on environment variables, see the *IBM Informix Guide to SQL: Reference*.

Examples of Client/Server Configurations

The next several sections show the correct entries in the **sqlhosts** file or SQLHOSTS registry key for several client/server connections. The following examples are included:

- Using a shared-memory connection
- Using a local-loopback connection
- Using a network connection
- Using multiple connection types
- Accessing multiple database servers



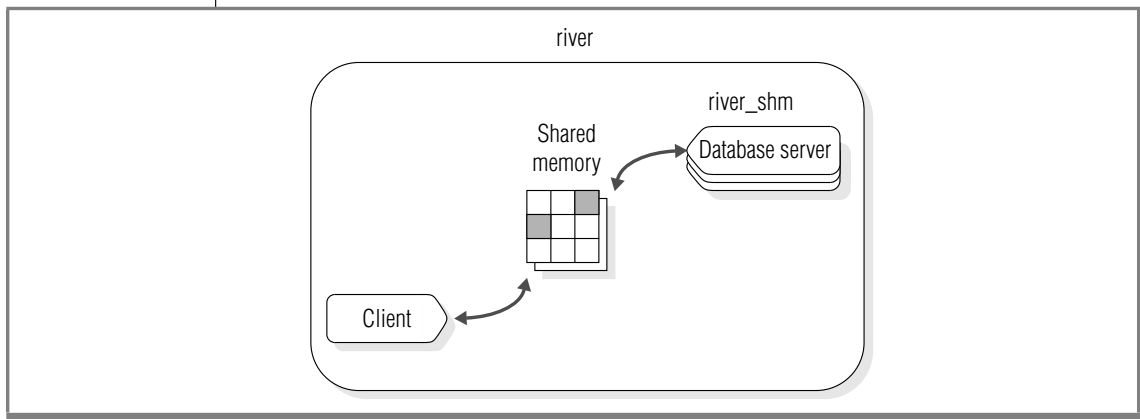
Important: In the following examples, you can assume that the network-configuration files *hosts* and *services* have been correctly prepared even if they are not explicitly mentioned.

UNIX

Using a Shared-Memory Connection

Figure 3-13 shows a shared-memory connection on the computer named **river**.

Figure 3-13
A Shared-Memory Connection



The ONCONFIG configuration file for this installation includes the following line:

```
DBSERVERNAME river_shm
```

Figure 3-14 shows a correct entry for the **sqlhosts** file or SQLHOSTS registry key.

Figure 3-14
sqlhosts entry

dbservername	nettype	hostname	servicename
river_shm	onipcshm	river	rivershm

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

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

Because this is a shared-memory connection, no entries in network configuration files are required. For a shared-memory connection, you can choose arbitrary values for the **hostname** and **servicename** fields of the **sqlhosts** file or SQLHOSTS registry key.

For more information about shared-memory connections, refer to [“How a Client Attaches to the Communications Portion”](#) on page 7-11.

Using a Local-Loopback Connection

[Figure 3-15](#) shows a local-loopback connection that uses sockets and TCP/IP. The name of the host computer is **river**.

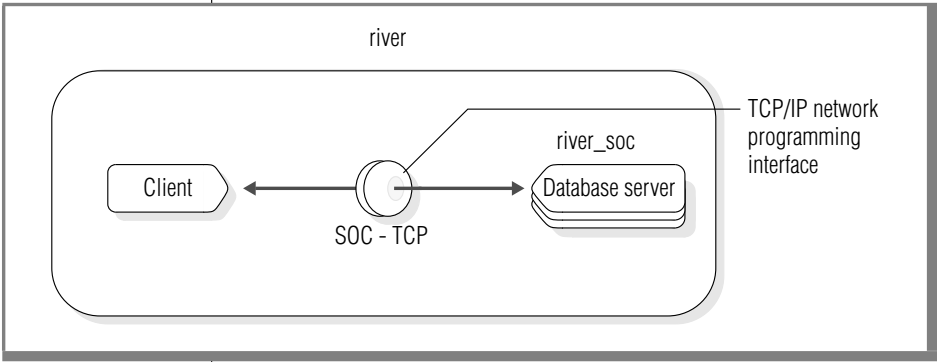


Figure 3-15
Local-Loopback Connection

[Figure 3-16](#) shows the correct entry for the **sqlhosts** file or SQLHOSTS registry key.

Figure 3-16
sqlhosts entry

dbservername	nettype	hostname	servicename
river_soc	onsoctcp	river	riverol

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

The ONCONFIG file includes the following line:

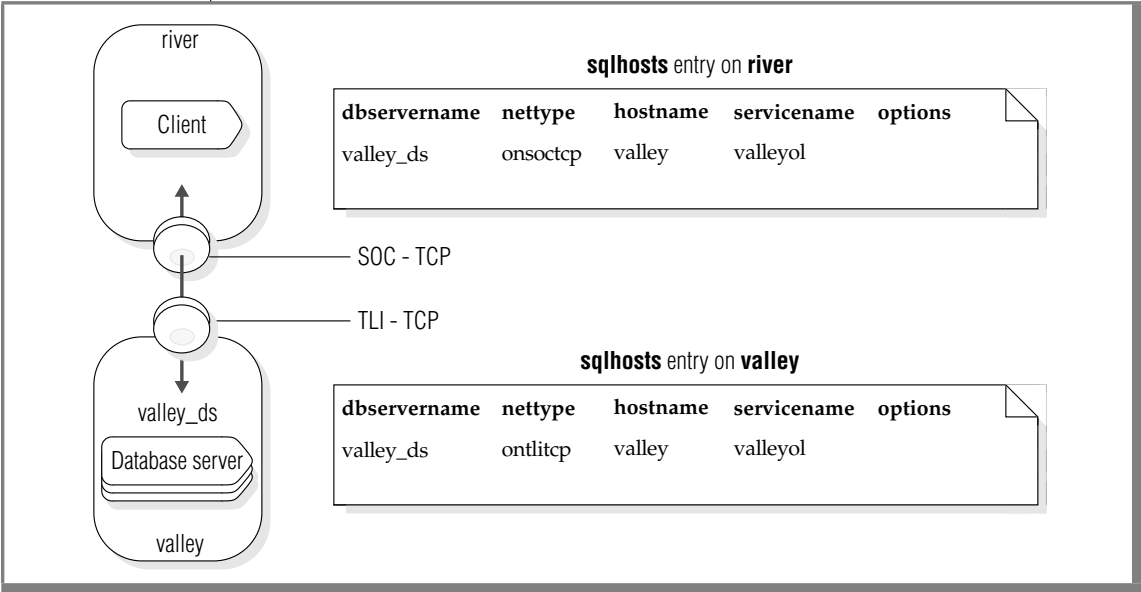
```
DBSERVERNAME river_soc
```

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

Using a Network Connection

Figure 3-17 shows a configuration in which the client application resides on host **river** and the database server resides on host **valley**.

Figure 3-17
A Network Configuration



An entry for the **valley_ds** database server is in the **sqlhosts** files or registries on both computers. Each entry in the **sqlhosts** file or SQLHOSTS registry key on the computer where the database server resides has a corresponding entry on the computer where the client application resides.

UNIX

Both computers are on the same TCP/IP network, but the host **river** uses sockets for its network programming interface, while the host **valley** uses TLI for its network programming interface. The **nettype** field must reflect the type of network programming interface used by the computer on which **sqlhosts** resides. In this example, the **nettype** field for the **valley_ds** database server on host **river** is **onsoctcp**, and the **nettype** field for the **valley_ds** database server on host **valley** is **ontlitcp**. ♦

UNIX

The sqlhosts File Entry for IPX/SPX

IPX/SPX software frequently provides TLI. [Figure 3-18](#) shows the entries in the **sqlhosts** file on both computers when the configuration in [Figure 3-17 on page 3-59](#) uses IPX/SPX instead of TCP/IP.

Figure 3-18
sqlhosts entry

dbservername	nettype	hostname	servicename
valley_us	ontlisp	valley_nw	valley_us

In this case, the **hostname** field contains the name of the NetWare file server. The **servicename** field contains a name that is unique on the IPX/SPX network and is the same as the **dbservername**.

Using Multiple Connection Types

A single instance of the database server can provide more than one type of connection. [Figure 3-19](#) illustrates such a configuration. The database server is on host **river**. Client A connects to the database server with a shared-memory connection because shared memory is fast. Client B must use a network connection because the client and server are on different computers.

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

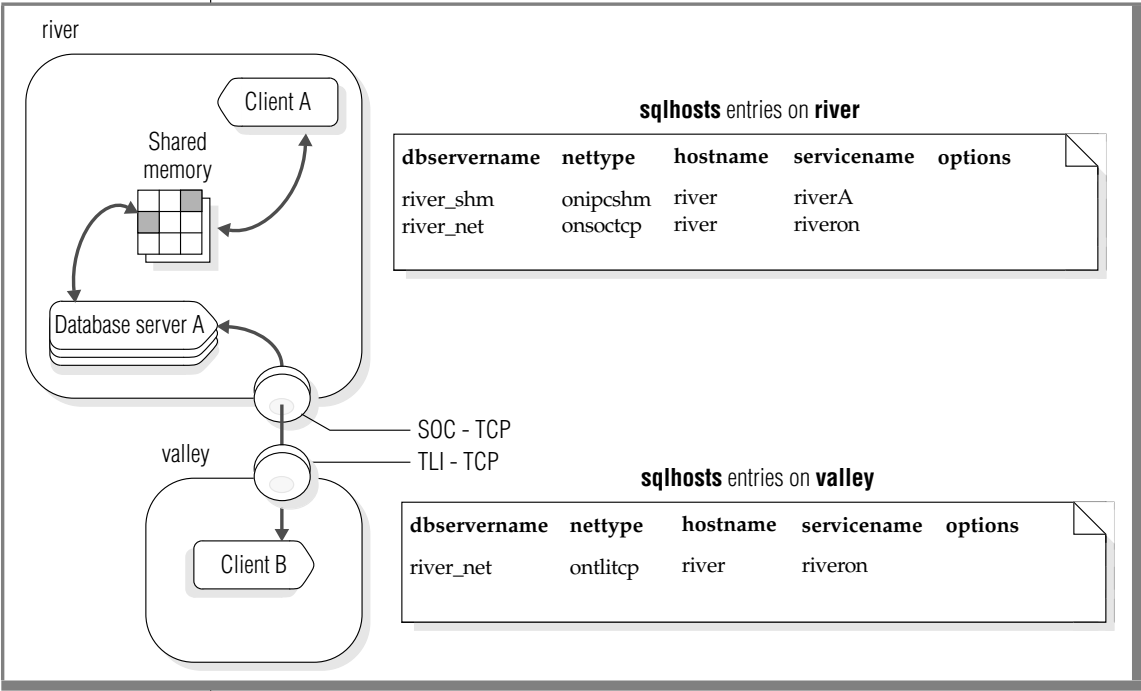
- Put DBSERVERNAME and DBSERVERALIASES entries in the ONCONFIG configuration file.
- Put an entry in the **sqlhosts** file or SQLHOSTS registry key for each database server/connection type pair.

For the configuration in [Figure 3-19](#), the database server has two dbserver-names: **river_net** and **river_shm**. The ONCONFIG configuration file includes the following entries:

```
DBSERVERNAME    river_net
DBSERVERALIASESriver_shm
```

Figure 3-19

A UNIX Configuration That Uses Multiple Connection Types



The dbservername used by a client application determines the type of connection that is used. Client A uses the following statement to connect to the database server:

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

In the **sqlhosts** file or SQLHOSTS registry key, the **nettype** associated with the name **river_shm** specifies a shared-memory connection, so this connection is a shared-memory connection.

Client B uses the following statement to connect to the database server:

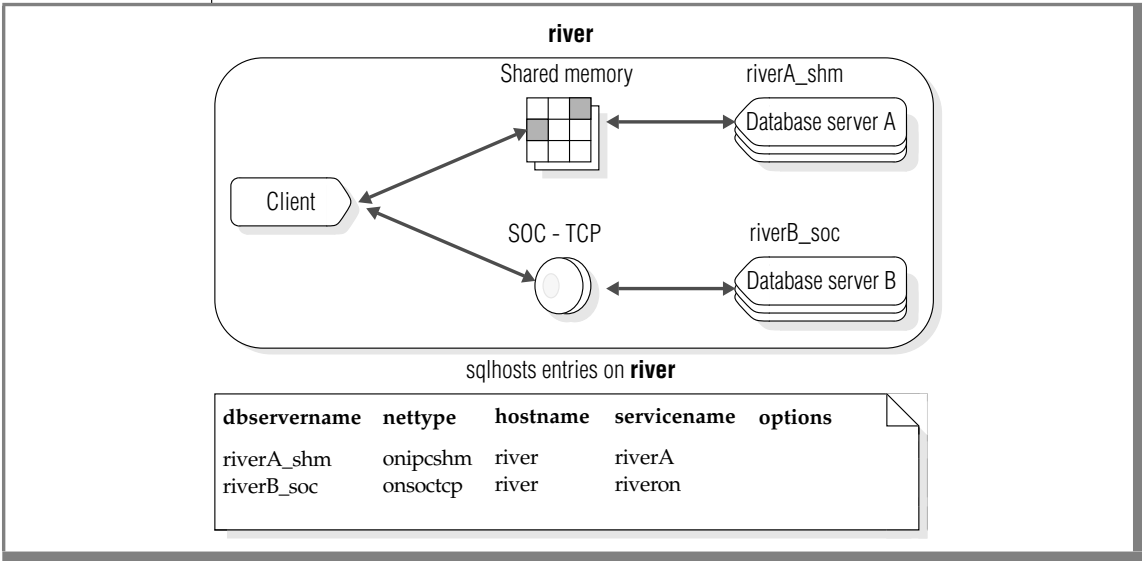
```
CONNECT TO '@river_net'
```

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

Accessing Multiple Database Servers

Figure 3-20 shows a configuration with two database servers on host **river**. When more than one database server is active on one computer, it is known as *multiple residency*. (For more information about multiple residency, see your *Installation Guide*.)

Figure 3-20
Multiple Database Servers on UNIX



For the configuration in Figure 3-20, you must prepare two ONCONFIG configuration files, one for database server **A** and the other for database server **B**. The **sqlhosts** file or SQLHOSTS registry key includes the connectivity information for both database servers.

The **ONCONFIG** configuration file for database server **A** includes the following line:

```
DBSERVERNAMEriverA_shm
```

The **ONCONFIG** configuration file for database server **B** includes the following line:

```
DBSERVERNAMEriverB_soc
```

Using IBM Informix MaxConnect

IBM Informix MaxConnect is a networking product for Informix database server environments on UNIX. MaxConnect manages large numbers (from several hundred to tens of thousands) of client/server connections. MaxConnect multiplexes connections so that the ratio of client connections to database connections can be 200:1 or higher. MaxConnect increases system scalability to many thousands of connections and saves system resources, reducing response times and CPU requirements. MaxConnect is best for OLTP data transfers and not recommended for large multimedia data transfers.

Install MaxConnect separately from your Informix database server and client applications. For maximum performance benefit, install MaxConnect either on a separate computer to which Informix clients connect or on the client application server. You can install MaxConnect in the following configurations:

- On a dedicated server to which Informix clients connect
- On the client application server
- On the database server computer

Two protocols for multiplexing connections, **ontliimc** and **onsocimc**, are available for MaxConnect users. You can use the **ontliimc** and **onsocimc** protocols in the following two configurations:

- To connect MaxConnect to the database server.
In this configuration, the client connections are multiplexed and use packet aggregation.



- To connect the client applications directly to the database server without going through MaxConnect.

In this configuration, the client does not get the benefits of connection multiplexing or packet aggregation. Choose this configuration when the client application is transferring simple- or smart-large-object data, because a direct connection to the database server is best.

For more information on how to configure MaxConnect and monitor it with the **onstat -g imc** and **imcadmin** commands, see the *IBM Informix MaxConnect*.

Important: *MaxConnect and the “IBM Informix MaxConnect” ship separately from the Informix database server.*

Initializing the Database Server

In This Chapter	4-3
Types of Initialization	4-3
Initializing Disk Space.	4-4
Initialization Steps	4-5
Process Configuration File	4-6
Create Shared-Memory Portions	4-7
Initialize Shared-Memory	4-8
Initialize Disk Space	4-8
Start All Required Virtual Processors	4-8
Make Necessary Conversions	4-9
Initiate Fast Recovery.	4-9
Initiate a Checkpoint	4-9
Document Configuration Changes	4-9
Create the oncfg_servername.servernum File	4-10
Drop Temporary Tblspaces.	4-10
Set Forced Residency If Specified	4-10
Return Control to User	4-11
Create sysmaster Database and Prepare SMI Tables	4-11
Create the sysutils Database	4-12
Monitor Maximum Number of User Connections	4-12
Database Server Operating Modes	4-12
Changing Database Server Operating Modes	4-14
Users Permitted to Change Modes	4-14
ISA Options for Changing Modes	4-15
On-Monitor Options for Changing Modes	4-16

Command-Line Options for Changing Modes	4-16
From Offline to Quiescent	4-16
From Offline to Online	4-17
From Quiescent to Online	4-17
Gracefully from Online to Quiescent	4-17
Immediately from Online to Quiescent	4-18
From Any Mode Immediately to Offline	4-19

In This Chapter

This chapter describes how to initialize the database server and change database server operating modes, and the activities that take place during initialization.

Types of Initialization

Initialization of the database server refers to two related activities: shared-memory initialization and disk-space initialization.

Shared-memory initialization or bringing up or starting the server establishes the contents of database server shared memory, as follows: internal tables, buffers, and the shared-memory communication area. Shared memory is initialized every time the database server starts up.

Two key differences distinguish shared-memory initialization from disk-space initialization:

- Shared-memory initialization has no effect on disk-space allocation or layout. No data is destroyed.
- Shared-memory initialization performs fast recovery.

Disk-space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk. When you initialize disk space, the database server automatically initializes shared memory as part of the process. Disk space is initialized the first time the database server starts up. It is only initialized thereafter during a cold restore or at the request of the database server administrator.

Warning: When you initialize disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all the data in the earlier database server becomes inaccessible and, in effect, is destroyed.



Initializing Disk Space

The database server must be in offline mode when you begin initialization.

If you are starting a database server for the first time or you want to remove all dbspaces and their associated data, use the following methods to initialize the disk space and to bring the database server into online mode.

Operating System	Action
UNIX	You must be logged in as informix or root to initialize the database server. Execute oninit -iy .
Windows	<div>You must be a member of the Administrators or Power Users group to initialize the database server.<ul style="list-style-type: none">■ The database server runs as a service. In the Services control application, choose the database server service and type -iy in the Startup parameters field. Then click Start.■ On the command line, use the starts dbservername -iy command.</div>

Warning: When you execute these commands, all existing data in the database server disk space is destroyed. Use the **-i** flag only when you are starting a new instance of the database server.



Windows

When you install the database server and choose to initialize a new instance of the database server, or when you use the instance manager program to create a new instance of the database server, the database server is initialized for you.

It is recommended that you do not use the **oninit -iy** command to initialize the database server unless you are diagnosing a problem. ♦

For more information about **oninit**, refer to the utilities chapter in the *Administrator's Reference*.

For information on using these utilities to initialize the database server and change the database server mode, see [“Starting the Database Server and Initializing Disk Space” on page 1-22](#).

Initialization Steps

Disk-space initialization always includes the initialization of shared memory. However, some activities that normally take place during shared-memory initialization, such as recording configuration changes, are not required during disk initialization because those activities are not relevant with a newly initialized disk.

Figure 4-1 shows the main tasks completed during the two types of initialization. The following sections discuss each step.

Figure 4-1
Initialization Steps

Shared-Memory Initialization	Disk Initialization
Process configuration file.	Process configuration file.
Create shared-memory segments.	Create shared-memory segments.
Initialize shared-memory structures.	Initialize shared-memory structures.
	Initialize disk space.
Start all required virtual processors.	Start all required virtual processors.
Make necessary conversions.	
Initiate fast recovery.	
Initiate a checkpoint.	Initiate a checkpoint.
Document configuration changes.	
Update <code>oncfg_servername.servernum</code> file.	Update <code>oncfg_servername.servernum</code> file.
Change to quiescent mode.	Change to quiescent mode.
Drop temporary tblspaces (optional).	
Set forced residency, if requested.	Set forced residency, if specified.
Change to online mode and return control to user.	Change to online mode and return control to user.

(1 of 2)

Shared-Memory Initialization	Disk Initialization
If the SMI tables are not current, update the tables.	Create sysmaster database that contains the SMI tables.
	Create the sysutils database.
Monitor maximum number of user connections at each checkpoint.	Monitor maximum number of user connections at each checkpoint.

(2 of 2)

Process Configuration File

The database server uses configuration parameters to allocate shared-memory segments during initialization. If you modify a shared-memory configuration parameter, you must shut down and restart the database server for the change to take effect.

The **ONCONFIG**, **onconfig**, and **onconfig.std** files are stored in **\$INFORMIXDIR/etc** on UNIX and **%INFORMIXDIR%\etc** on Windows. During initialization, the database server looks for configuration values in the following files, in this order:

1. If the **ONCONFIG** environment variable is set, the database server reads values from the **ONCONFIG** file.
If the **ONCONFIG** environment variable is set, but the database server cannot access the specified file, it returns an error message.
2. If the **ONCONFIG** environment variable is not set, the database server reads the configuration values from the **onconfig** file.
3. If you omit a configuration parameter in your **ONCONFIG** file, the database server reads the configuration values from the **\$INFORMIXDIR/etc/onconfig.std** file.

It is recommended that you *always* set the **ONCONFIG** environment variable before you initialize the database server. The default configuration files are intended as templates and not as functional configurations. However, the server will not initialize if **onconfig_std** is missing. For more information about the configuration file, refer to [“Configuring the Database Server” on page 1-18](#).

The initialization process compares the values in the current configuration file with the previous values, if any, that are stored in the root dbspace reserved page, PAGE_CONFIG. When differences exist, the database server uses the values from the current ONCONFIG configuration file for initialization.

Create Shared-Memory Portions

The database server uses the configuration values to calculate the required size of the database server resident shared memory. In addition, the database server computes additional configuration requirements from internal values. Space requirements for overhead are calculated and stored.

To create shared memory, the database server acquires the shared-memory space from the operating system for three different types of memory:

- Resident portion, used for data buffers and internal tables
- Virtual portion, used for most system and user-session memory requirements
- IPC communication portion, used for IPC communication

The database server allocates this portion of shared memory only if you configure an IPC shared-memory connection. ♦

Next, the database server attaches shared-memory segments to its virtual address space and initializes shared-memory structures. For more information about shared-memory structures, refer to [“Virtual Portion of Shared Memory” on page 7-22](#).

After initialization is complete and the database server is running, it can create additional shared-memory segments as needed. The database server creates segments in increments of the page size.

Initialize Shared-Memory

After the database server attaches to shared memory, it clears the shared-memory space of uninitialized data. Next the database server lays out the shared-memory header information and initializes data in the shared-memory structures. The database server lays out the space needed for the logical-log buffer, initializes the structures, and links together the three individual buffers that form the logical-log buffer. For more information about these structures, refer to the **onstat** utility in the *Administrator's Reference*.

After the database server remaps the shared-memory space, it registers the new starting addresses and sizes of each structure in the new shared-memory header.

During shared-memory initialization, disk structures and disk layout are not affected. The database server reads essential address information, such as the locations of the logical and physical logs, from disk and uses this information to update pointers in shared memory.

Initialize Disk Space

This procedure is performed only during disk-space initialization. After shared-memory structures are initialized, the database server begins initializing the disk. The database server initializes all the reserved pages that it maintains in the root dbspace on disk and writes PAGE_PZERO control information to the disk.

Start All Required Virtual Processors

The database server starts all the virtual processors that it needs. The parameters in the ONCONFIG file influence what processors are started. For example, the NETTYPE parameter can influence the number and type of processors started for making connections. For more information about virtual processors, refer to [“Virtual Processors” on page 5-3](#).

Make Necessary Conversions

The database server checks its internal files. If the files are from an earlier version, it updates these files to the current format. For information about database conversion, refer to the *IBM Informix Migration Guide*.

Initiate Fast Recovery

The database server checks if fast recovery is needed and, if so, initiates it. For more information about fast recovery, refer to [“Fast Recovery” on page 15-18](#).

Fast recovery is not performed during disk-space initialization because there is not yet anything to recover.

Initiate a Checkpoint

After fast recovery executes, the database server initiates a full checkpoint. As part of the checkpoint procedure, the database server writes a checkpoint-complete message in the message log. For more information about checkpoints, refer to [“Checkpoints” on page 15-11](#).

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

Document Configuration Changes

The database server compares the current values stored in the configuration file with the values previously stored in the root dbspace reserved page PAGE_CONFIG. When differences exist, the database server notes both values (old and new) in a message to the message log.

This task is not performed during disk-space initialization.

Create the `oncfg_servername.servernum` File

The database server creates the `oncfg_servername.servernum` file and updates it every time that you add or delete a dbspace, blobspace, logical-log file, or chunk. You do not need to manipulate this file in any way, but you can see it listed in your `$INFORMIXDIR/etc` directory on UNIX or in your `%INFORMIXDIR%\etc` directory on Windows. The database server uses this file during a full-system restore.

For more information about the `oncfg_servername.servernum` file, refer to the section on files that the database server uses in the *Administrator's Reference*.

Drop Temporary Tblspaces

The database server searches through all dbspaces for temporary tblspaces. (If you use the `-p` option of `oninit` to initialize the database server, the database server skips this step.) These temporary tblspaces, if any, are tblspaces left by user processes that died prematurely and were unable to perform proper cleanup. The database server deletes any temporary tblspaces and reclaims the disk space. For more information about temporary tblspaces, refer to [“Temporary Tables” on page 9-42](#).

This task is not performed during disk-space initialization.

Set Forced Residency If Specified

If the value of the `RESIDENT` configuration parameter is `-1` or a number greater than `0`, the database server tries to enforce residency of shared memory. If the host computer system does not support forced residency, the initialization procedure continues. Residency is not enforced, and the database server sends an error message to the message log. For more information about the `RESIDENT` configuration parameter, refer to the *Administrator's Reference*.

Return Control to User

The database server writes the `IBM Informix Dynamic Server initialized - complete disk initialized` message in the message log. The database server also dynamically allocates a virtual shared-memory segment.

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed in the following locations:

- The command line
- The database server message log file, specified by the `MSGPATH` configuration parameter
For more information about the `MSGPATH` parameter, refer to the *Administrator's Reference*.
- **Summary** section of IBM Informix Server Administrator (ISA)

Create **sysmaster** Database and Prepare SMI Tables

Even though the database server has returned control to the user, it has not finished its work. The database server now checks the *system-monitoring interface* (SMI) tables. If the SMI tables are not current, the database server updates the tables. If the SMI tables are not present, as is the case when the disk is initialized, the database server creates the tables. After the database server builds the SMI tables, it puts the message `sysmaster database built successfully` in the message log. The database server also re-creates the **sysmaster** database during conversion and reversion. For more information about SMI tables, refer to the chapter on the **sysmaster** database in the *Administrator's Reference*.

If you shut down the database server before it finishes building the SMI tables, the process of building the tables aborts. This condition does not damage the database server. The database server simply builds the SMI tables the next time that you bring the database server online. However, if you do not allow the SMI tables to finish building, you cannot run any queries against those tables, and you cannot use ON-Bar for backups.

After the SMI tables have been created, the database server is ready for use. The database server runs until you stop it or, possibly, until a malfunction. It is recommended that you *do not* try to stop the database server by killing a virtual processor or another database server process. For more information, refer to [“Starting and Stopping Virtual Processors” on page 6-6](#).

Create the sysutils Database

The database server drops and re-creates the **sysutils** database during disk initialization, conversion, or reversion. ON-Bar stores backup and restore information in the **sysutils** database. Wait until the message `sysutils database built successfully` displays in the message log. For more information, see the *IBM Informix Backup and Restore Guide*.

Monitor Maximum Number of User Connections

At each checkpoint, the database server prints the maximum number of user connections in the message `log:maximum server connections number`. You can monitor the number of users who have connected to the database server since the last restart or disk initialization.

This message helps the customer track license usage to determine when to purchase more licenses. The number displayed is reset when the customer reinitializes the database server.

Database Server Operating Modes

You can determine the current database server mode by executing the **onstat** utility from the command line. The **onstat** header displays the mode.

The database server has three principal modes of operation, as [Figure 4-2](#) illustrates.

Figure 4-2
Operating Modes

Operating Mode	Description
Offline mode	When the database server is not running. No shared memory is allocated.
Quiescent, or administration, mode	When the database server processes are running and shared-memory resources are allocated, but the system does not allow database user access. Only the administrator (user informix) can access the database server.
Online mode	Users can connect with the database server and perform all database activities. This is the normal operating mode of the database server. User informix or user root can use the command-line utilities to change many database server ONCONFIG parameter values while the database server is online.
Maintenance mode	Online mode with access restricted to users root and informix . Users cannot initiate maintenance mode. The database server uses this mode.

In addition, the database server can also be in one of the following modes:

- *Read-only mode* is used by the secondary database server in an HDR pair. An application can query a database server that is in read-only mode, but the application cannot write to a read-only database.
- *Recovery mode* is transitory. It occurs when the database server performs fast recovery or recovers from a system archive or system restore. Recovery occurs during the change from offline to quiescent mode.
- *Shutdown mode* is transitory. It occurs when the database server is moving from online to quiescent mode or from online (or quiescent) to offline mode. Current users access the system, but no new users are allowed access.

Once shutdown mode is initiated, it cannot be cancelled.

Windows

Changing Database Server Operating Modes

This section describes how to change from one database server operating mode to another with the **oninit** and **onmode** utilities and with ISA.

In Windows, the database server runs as a service. Windows provides a service control application (also called the Services tool) to start, stop, and pause a service. The service control application is located in the Control Panel program group. The service name for the database server includes the database server name (the value of DBSERVERNAME in the ONCONFIG file). For example, the Dynamic Server service for the newyork database server is as follows:

```
IBM Informix Database Server - newyork
```

To change mode with the Services tool, start the tool and select the database server service. Then choose the appropriate button in the Services window. The tables shown later in this chapter explain which button you select for each mode.

To start and stop the database server, you can use other Windows tools, such as the NET command and the Server Manager tool. For more information about these methods, consult your Windows operating-system documentation. ♦



Tip: After you change the mode of your database server, execute the **onstat** command to verify the current server status.

Users Permitted to Change Modes

Only users who are logged in as **root** or **informix** can change the operating mode of the database server. ♦

Figure 4-2 on page 4-13 shows which users can change the operating mode of the database server in Windows. If you use ISA, you can log in to the operating system as any user but you must log in to Apache as user **informix**. The Apache server runs as a member of the **Informix-Admin** group.

UNIX

Windows

Figure 4-3
Changing Operating Modes in Windows

Changing Operating Mode	Administrators Group	Informix-Admin Group
command-line utilities such as starts		✓
ISA		✓
Services control panel	✓	

◆

ISA Options for Changing Modes

You can use ISA to change the database server mode. For more information, see the ISA online help.

Action	ISA Option
Initialize the database server.	Mode→Online (Initialize Disks) or Quiescent (Initialize Disks)
Change from offline to quiescent.	Mode→Quiescent
Change from offline or quiescent to online.	Mode→Online
Change gracefully from online to quiescent.	Mode→Quiescent (Graceful)
Change immediately from online to quiescent.	Mode→Quiescent (Immediate)
Shut down the database server.	Mode→Offline

UNIX

On-Monitor Options for Changing Modes

You can use ON-Monitor to change the database server mode on UNIX. For more information, see the section on ON-Monitor in the *Administrator's Reference*.

Action	ISA Option
Initialize the database server.	Parameters→Initialize
Change from offline to quiescent.	Mode→Startup
Change from offline or quiescent to online.	Mode→Online
Change gracefully from online to quiescent.	Mode→Graceful Shutdown
Change immediately from online to quiescent.	Mode→Immediate Shutdown
Shut down the database server.	Mode→Take Offline

Command-Line Options for Changing Modes

You can use these command-line options to change the database server mode.

From Offline to Quiescent

When the database server changes from offline mode to quiescent mode, the database server initializes shared memory.

When the database server is in quiescent mode, no sessions can gain access to the database server. In quiescent mode, any user can see status information.

Operating System	Action
UNIX	■ Execute oninit -s .
Windows	■ On the command line, use the starts dbservername -s command.

From Offline to Online

When you take the database server from offline mode to online mode, the database server initializes shared memory. When the database server is in online mode, it is available to all database server sessions.

Operating System	Action
UNIX	■ Execute oninit .
Windows	■ With the Services tool, select the database server service and click Start. ■ On the command line, use the starts dbservername command.

From Quiescent to Online

When you take the database server from quiescent mode to online mode, all sessions gain access.

If you have already taken the database server from online mode to quiescent mode and you are now returning the database server to online mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.

Operating System	Action
UNIX	■ Execute onmode -m .
Windows	■ With the Services tool, choose the database server service and click Continue. ■ Execute onmode -m .

Gracefully from Online to Quiescent

Take the database server gracefully from online mode to quiescent mode to restrict access to the database server without interrupting current processing.

After you perform this task, the database server sets a flag that prevents new sessions from gaining access to the database server. Current sessions are allowed to finish processing.

Once you initiate the mode change, it cannot be cancelled. During the mode change from online to quiescent, the database server is considered to be in Shutdown mode.

Operating System	Action
UNIX	■ Execute onmode -s or onmode -sy .
Windows	■ Execute onmode -s or onmode -sy . ■ With the Services tool, choose the database server service and click Pause.

Immediately from Online to Quiescent

Take the database server immediately from online mode to quiescent mode to restrict access to the database server as soon as possible. Work in progress can be lost.

A prompt asks for confirmation of the immediate shutdown. If you confirm, the database server sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates the session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

The database server cleans up all sessions that the database server terminated. Active transactions are rolled back.

Operating System	Action
UNIX	■ Execute onmode -u or onmode -uy . The -y option eliminates the need to confirm the prompt.
Windows	■ Execute onmode -u or onmode -uy . The -y option eliminates the need to confirm the prompt.

From Any Mode Immediately to Offline

You can take the database server immediately from any mode to offline mode. After you take the database server to offline mode, restart the database server in quiescent or online mode. When you restart the database server, it performs a fast recovery to ensure that the data is logically consistent.

A prompt asks for confirmation to go offline. If you confirm, the database server initiates a checkpoint request and sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates this session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

After you take the database server to offline mode, restart the database server in quiescent or online mode. When you restart the database server, it performs a fast recovery to ensure that the data is logically consistent.

The database server cleans up all sessions that were terminated by the database server. Active transactions are rolled back.

Operating System	Action
UNIX	<ul style="list-style-type: none"> ■ Execute onmode -k or onmode -ky. The -y option eliminates the automatic prompt that confirms an immediate shutdown.
Windows	<ul style="list-style-type: none"> ■ Execute onmode -k or onmode -ky. The -y option eliminates the automatic prompt that confirms an immediate shutdown. ■ With the Services tool, choose the database server service and click Stop.

Disk, Memory, and Process Management

- Chapter 5** **Virtual Processors and Threads**
- Chapter 6** **Managing Virtual Processors**
- Chapter 7** **Shared Memory**
- Chapter 8** **Managing Shared Memory**
- Chapter 9** **Data Storage**
- Chapter 10** **Managing Disk Space**



Virtual Processors and Threads

In This Chapter	5-3
Virtual Processors	5-3
Threads	5-4
Types of Virtual Processors.	5-5
Advantages of Virtual Processors	5-9
Sharing Processing	5-9
Saving Memory and Resources	5-9
Processing in Parallel	5-10
Adding and Dropping Virtual Processors in Online Mode	5-11
Binding Virtual Processors to CPUs	5-12
How Virtual Processors Service Threads	5-12
Control Structures	5-13
Context Switching	5-13
Stacks	5-15
Queues	5-16
Ready Queues	5-17
Sleep Queues	5-17
Wait Queues	5-18
Mutexes	5-19
Virtual-Processor Classes.	5-19
CPU Virtual Processors	5-19
Determining the Number of CPU Virtual Processors Needed	5-20
Running on a Multiprocessor Computer	5-21
Running on a Single-Processor Computer	5-21
Adding and Dropping CPU Virtual Processors in Online Mode.	5-22
Preventing Priority Aging.	5-22
Using Processor Affinity	5-22

User-Defined Classes of Virtual Processors	5-24
Determining the Number of User-Defined Virtual Processors Needed.	5-24
Using User-Defined Virtual Processors	5-24
Specifying a User-Defined Virtual Processor	5-25
Assigning a UDR to a User-Defined Virtual-Processor Class	5-25
Adding and Dropping User-Defined Virtual Processors in Online Mode.	5-26
Java Virtual Processors	5-26
Disk I/O Virtual Processors	5-27
I/O Priorities	5-27
Logical-Log I/O	5-28
Physical-Log I/O	5-28
Asynchronous I/O	5-29
Network Virtual Processors.	5-31
Specifying Network Connections	5-32
Running Poll Threads on CPU or Network Virtual Processors	5-32
Specifying the Number of Networking Virtual Processors	5-33
Specifying Listen and Poll Threads for the Client/Server Connection	5-33
Starting Multiple Listen Threads	5-36
Communications Support Module Virtual Processor	5-38
Optical Virtual Processor	5-38
Audit Virtual Processor	5-38
Miscellaneous Virtual Processor	5-38

In This Chapter

This chapter explains how the database server uses virtual processors and threads within virtual processors to improve performance. It explains the types of virtual processors and how threads run within the virtual processors.

Virtual Processors

Database server processes are called *virtual processors* because the way they function is similar to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a database server virtual processor runs multiple *threads* to service multiple SQL client applications.

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

Figure 5-1 illustrates the relationship of client applications to virtual processors. A small number of virtual processors serve a much larger number of client applications or queries.

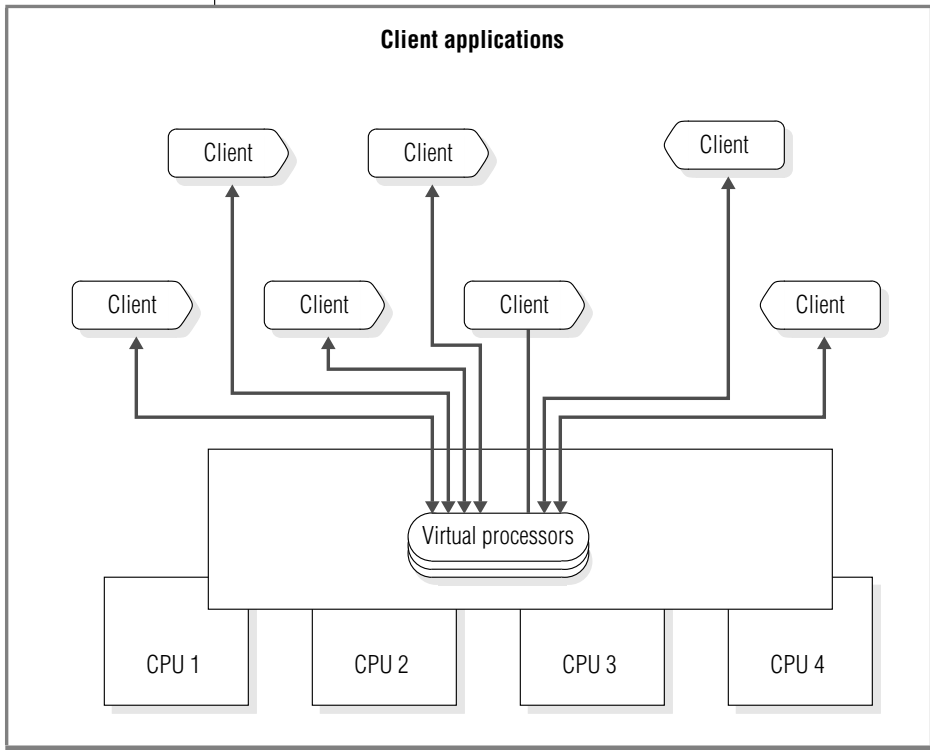


Figure 5-1
Virtual Processors

Threads

A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU. The virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that the virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes, but they make fewer demands on the operating system.

Database server virtual processors are *multithreaded* because they run multiple concurrent threads.

The nature of threads is as follows.

Operating System	Description of Thread
UNIX	A thread is a task that the virtual processor schedules internally for processing.
Windows	A thread is a task that the virtual processor schedules internally for processing. Because the virtual processor is implemented as a Windows thread, database server threads run within Windows threads.



Important: Throughout this chapter, all references to “thread” refer to the threads created, scheduled, and destroyed by the database server. All references to “Windows threads” refer to the threads created, scheduled, and destroyed by Windows.

A virtual processor runs threads on behalf of SQL client applications (*session threads*) and also to satisfy internal requirements (*internal threads*). In most cases, for each connection by a client application, the database server runs one session thread. The database server runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks. For cases in which the database server runs multiple session threads for a single client, refer to [“Processing in Parallel” on page 5-10](#).

A *user thread* is a database server thread that services requests from client applications. User threads include session threads, called **sqlexec** threads, which are the primary threads that the database server runs to service client applications.

User threads also include a thread to service requests from the **onmode** utility, threads for recovery, B-tree scanner threads, and page-cleaner threads.

To display active user threads, use **onstat -u**. For more information on monitoring sessions and threads, refer to your *Performance Guide*.

Types of Virtual Processors

[Figure 5-2](#) shows the *classes* of virtual processors and the types of processing that they do. Each class of virtual processor is dedicated to processing certain types of threads.

The number of virtual processors of each class that you configure depends on the availability of physical processors (CPUs), hardware memory, and the database applications in use.

Figure 5-2
Virtual-Processor Classes

Virtual-Processor Class	Category	Purpose
CPU	Central processing	Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O (KAIO) where available. Can run a single poll thread, depending on configuration.
PIO	Disk I/O	Writes to the physical-log file (internal class) if it is in cooked disk space.
LIO	Disk I/O	Writes to the logical-log files (internal class) if they are in cooked disk space.
AIO	Disk I/O	Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
SHM	Network	Performs shared memory communication.
TLI	Network	Uses the Transport Layer Interface (TLI) to perform network communication.
SOC	Network	Uses sockets to perform network communication.
OPT (UNIX)	Optical	Performs I/O to optical disk.
ADM	Administrative	Performs administrative functions.
ADT	Auditing	Performs auditing functions.
MSC	Miscellaneous	Services requests for system calls that require a very large stack.

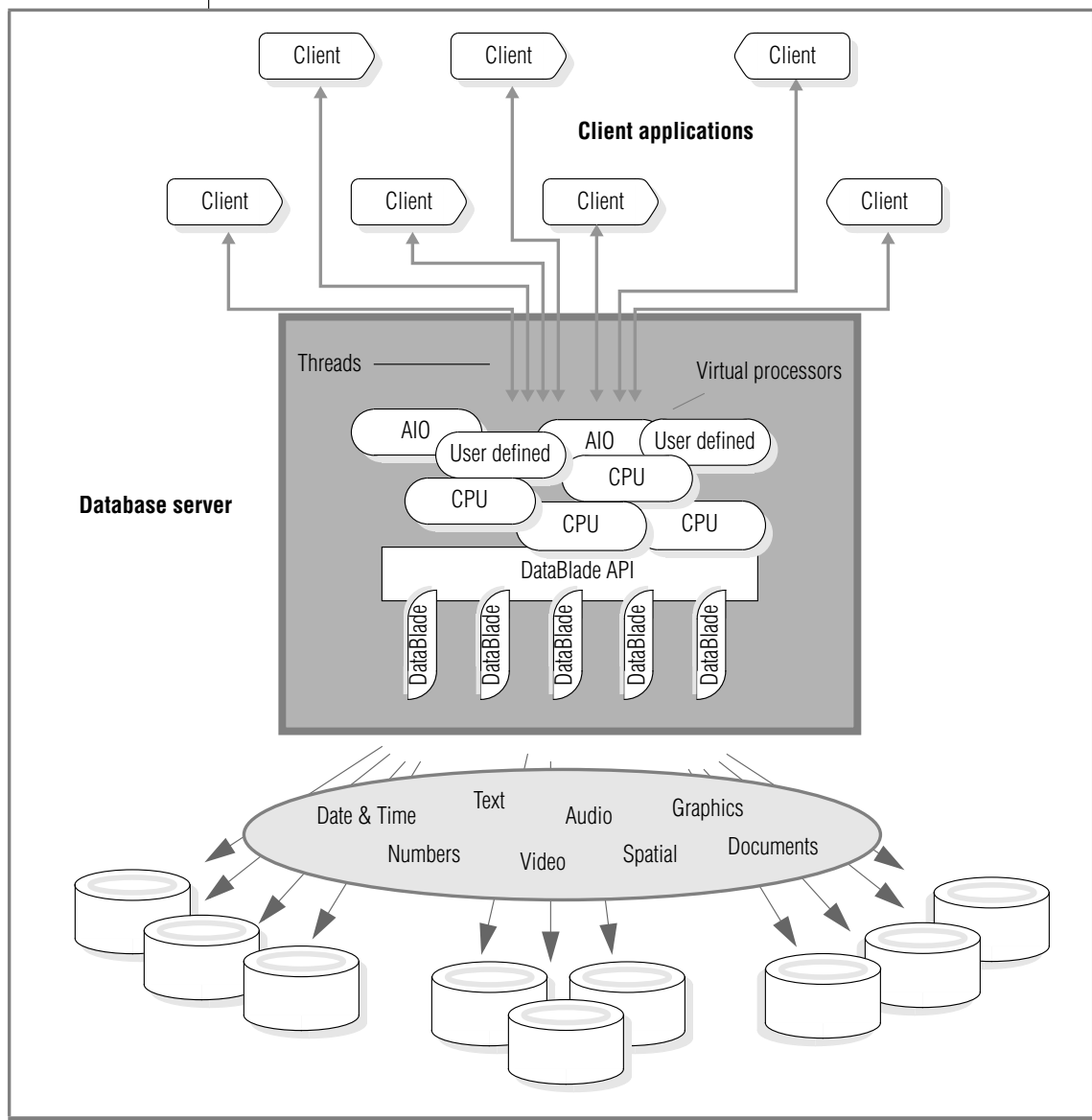
(1 of 2)

Virtual-Processor Class	Category	Purpose
CSM	Communications Support Module	Performs communications support service operations.
<i>classname</i>	User defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected. Specified with the VPCLASS configuration parameter. You must specify <i>classname</i> .
Java VP (JVP)	Java UDR	Executes Java UDRs. Contains the Java Virtual Machine (JVM).

(2 of 2)

Figure 5-3 illustrates the major components and the extensibility of the database server.

Figure 5-3
Database Server



Advantages of Virtual Processors

Compared to a database server process that services a single client application, the dynamic, multithreaded nature of a database server virtual processor provides the following advantages:

- Virtual processors can share processing.
- Virtual processors save memory and resources.
- Virtual processors can perform parallel processing.
- You can start additional virtual processors and terminate active CPU virtual processors while the database server is running.
- You can bind virtual processors to CPUs.

The following sections describe these advantages.

Sharing Processing

Virtual processors in the same class have identical code and share access to both data and processing queues in memory. Any virtual processor in a class can run any thread that belongs to that class.

Generally, the database server tries to keep a thread running on the same virtual processor because moving it to a different virtual processor can require some data from the memory of the processor to be transferred on the bus. When a thread is waiting to run, however, the database server can migrate the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the data.

Shared processing within a class of virtual processors occurs automatically and is transparent to the database user.

Saving Memory and Resources

The database server is able to service a large number of clients with a small number of server processes compared to architectures that have one client process to one server process. It does so by running a thread, rather than a process, for each client.

Multithreading permits more efficient use of the operating-system resources because threads share the resources allocated to the virtual processor. All threads that a virtual processor runs have the same access to the virtual-processor memory, communication ports, and files. The virtual processor coordinates access to resources by the threads. Individual processes, on the other hand, each have a distinct set of resources, and when multiple processes require access to the same resources, the operating system must coordinate the access.

Generally, a virtual processor can switch from one thread to another faster than the operating system can switch from one process to another. When the operating system switches between processes, it must stop one process from running on the processor, save its current processing state (or context), and start another process. Both processes must enter and exit the operating-system kernel, and the contents of portions of physical memory might need to be replaced. Threads, on the other hand, share the same virtual memory and file descriptors. When a virtual processor switches from one thread to another, the switch is simply from one path of execution to another. The virtual processor, which is a process, continues to run on the CPU without interruption. For a description of how a virtual processor switches from one thread to another, refer to [“Context Switching” on page 5-13](#).

Processing in Parallel

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

- Index building
- Sorting
- Recovery
- Scanning
- Joining
- Aggregation
- Grouping
- User-defined-routine (UDR) execution

For more information on parallel UDR execution, refer to *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Figure 5-4 illustrates parallel processing. When a client initiates index building, sorting, or logical recovery, the database server spawns multiple threads to work on the task in parallel, using as much of the computer resources as possible. While one thread is waiting for I/O, another can be working.

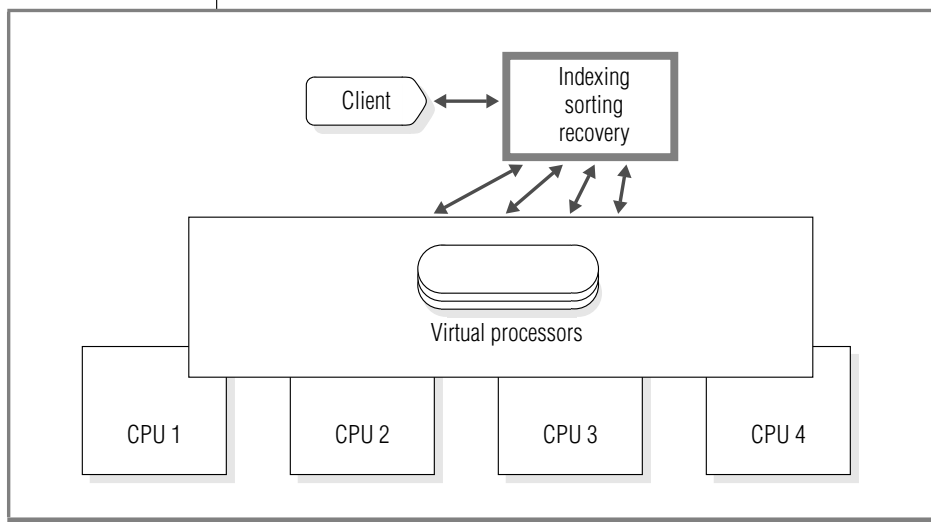


Figure 5-4
Parallel Processing

Adding and Dropping Virtual Processors in Online Mode

You can add virtual processors to meet increasing demands for service while the database server is running. For example, if the virtual processors of a class become compute bound or I/O bound (meaning that CPU work or I/O requests are accumulating faster than the current number of virtual processors can process them), you can start additional virtual processors for that class to distribute the processing load further.

You can add virtual processors for any of the classes while the database server is running. For more information, see [“Adding Virtual Processors in Online Mode” on page 6-7](#).

Windows

In Windows, you cannot drop a virtual processor of any class. ♦

While the database server is running, you can drop virtual processors of the CPU or a user-defined class. For more information, see [“Setting Virtual-Processor Configuration Parameters” on page 6-3](#).

Binding Virtual Processors to CPUs

Some multiprocessor systems allow you to bind a process to a particular CPU. This feature is called *processor affinity*.

On multiprocessor computers for which the database server supports processor affinity, you can bind CPU virtual processors to specific CPUs in the computer. When you bind a CPU virtual processor to a CPU, the virtual processor runs exclusively on that CPU. This operation improves the performance of the virtual processor because it reduces the amount of switching between processes that the operating system must do. Binding CPU virtual processors to specific CPUs also enables you to isolate database work on specific processors on the computer, leaving the remaining processors free for other work. Only CPU virtual processors can be bound to CPUs.

For information on how to assign CPU virtual processors to hardware processors, refer to [“Using Processor Affinity” on page 5-22](#).

How Virtual Processors Service Threads

At a given time, a virtual processor can run only one thread. A virtual processor services multiple threads concurrently by switching between them. A virtual processor runs a thread until it yields. When a thread yields, the virtual processor switches to the next thread that is ready to run. The virtual processor continues this process, eventually returning to the original thread when that thread is ready to continue. Some threads complete their work, and the virtual processor starts new threads to process new work. Because a virtual processor continually switches between threads, it can keep the CPU processing continually. The speed at which processing occurs produces the appearance that the virtual processor processes multiple tasks simultaneously and, in effect, it does.

Running multiple concurrent threads requires scheduling and synchronization to prevent one thread from interfering with the work of another. Virtual processors use the following structures and methods to coordinate concurrent processing by multiple threads:

- Control structures
- Context switching

- Stacks
- Queues
- Mutexes

This section describes how virtual processors use these structures and methods.

Control Structures

When a client connects to the database server, the database server creates a *session* structure, which is called a *session control block*, to hold information about the connection and the user. A session begins when a client connects to the database server, and it ends when the connection terminates.

Next, the database server creates a thread structure, which is called a *thread-control block* (TCB) for the session, and initiates a primary thread (**sqlexec**) to process the client request. When a thread *yields*—that is, when it pauses and allows another thread to run—the virtual processor saves information about the state of the thread in the thread-control block. This information includes the content of the process system registers, the program counter (address of the next instruction to execute), and the stack pointer. This information constitutes the *context* of the thread.

In most cases, the database server runs one primary thread per session. In cases where it performs parallel processing, however, it creates multiple session threads for a single client, and, likewise, multiple corresponding thread-control blocks.

Context Switching

A virtual processor switches from running one thread to running another one by *context switching*. The database server does not preempt a running thread, as the operating system does to a process, when a fixed amount of time (time-slice) expires. Instead, a thread yields at one of the following points:

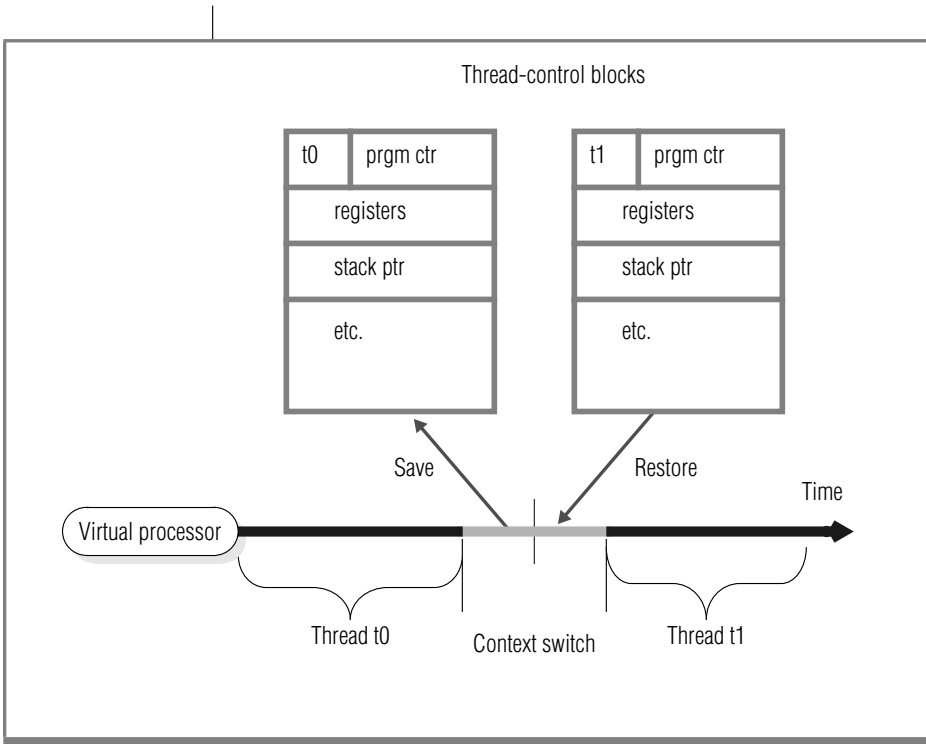
- A predetermined point in the code
- When the thread can no longer execute until some condition is met

When the amount of processing required to complete a task would cause other threads to wait for an undue length of time, a thread yields at a predetermined point. The code for such long-running tasks includes calls to the yield function at strategic points in the processing. When a thread performs one of these tasks, it yields when it encounters a yield function call. Other threads in the ready queue then get a chance to run. When the original thread next gets a turn, it resumes executing code at the point immediately after the call to the yield function. Predetermined calls to the yield function allow the database server to interrupt threads at points that are most advantageous for performance.

A thread also yields when it can no longer continue its task until some condition occurs. For example, a thread yields when it is waiting for disk I/O to complete, when it is waiting for data from the client, or when it is waiting for a lock or other resource.

When a thread yields, the virtual processor saves its context in the thread-control block. Then the virtual processor selects a new thread to run from a queue of ready threads, loads the context of the new thread from its thread-control block, and begins executing at the new address in the program counter. [Figure 5-5](#) illustrates how a virtual processor accomplishes a context switch.

Figure 5-5
Context Switch:
*How a Virtual
 Processor Switches
 from One Thread to
 Another*



Stacks

The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*. For information on how to set the size of the stack, refer to [“Stacks” on page 7-29](#).

The stack enables a virtual processor to protect the nonshared data of a thread from being overwritten by other threads that concurrently execute the same code. For example, if several client applications concurrently perform SELECT statements, the session threads for each client execute many of the same functions in the code. If a thread did not have a private stack, one thread could overwrite local data that belongs to another thread within a function.

When a virtual processor switches to a new thread, it loads a stack pointer for that thread from a field in the thread-control block. The stack pointer stores the beginning address of the stack. The virtual processor can then specify offsets to the beginning address to access data within the stack. [Figure 5-6](#) illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

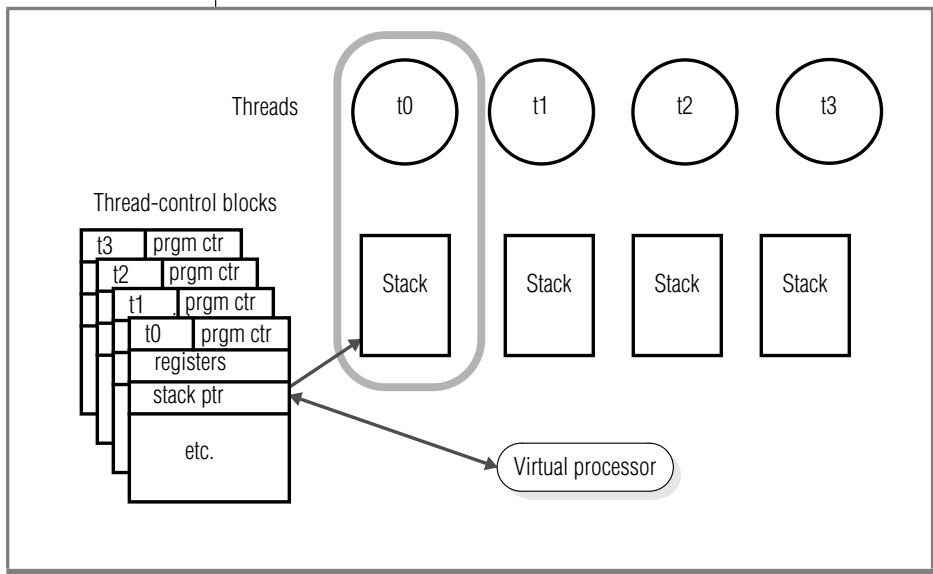


Figure 5-6
Virtual Processors
Segregate
Nonshared Data for
Each User

Queues

The database server uses three types of queues to schedule the processing of multiple, concurrently running threads:

- Ready queues
- Sleep queues
- Wait queues

Virtual processors of the same class share queues. This fact, in part, enables a thread to migrate from one virtual processor in a class to another when necessary.

Ready Queues

Ready queues hold threads that are ready to run when the current (running) thread yields. When a thread yields, the virtual processor picks the next thread with the appropriate priority from the ready queue. Within the queue, the virtual processor processes threads that have the same priority on a first-in-first-out (FIFO) basis.

On a multiprocessor computer, if you notice that threads are accumulating in the ready queue for a class of virtual processors (indicating that work is accumulating faster than the virtual processor can process it), you can start additional virtual processors of that class to distribute the processing load. For information on how to monitor the ready queues, refer to [“Monitoring Virtual Processors” on page 6-9](#). For information on how to add virtual processors while the database server is in online mode, refer to [“Adding Virtual Processors in Online Mode” on page 6-7](#).

Sleep Queues

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

The administration class (ADM) of virtual processors runs the system timer and special utility threads. Virtual processors in this class are created and run automatically. No configuration parameters impact this class of virtual processors.

The ADM virtual processor wakes up threads that have slept for the specified time. A thread that runs in the ADM virtual processor checks on sleeping threads at one-second intervals. If a sleeping thread has slept for its specified time, the ADM virtual processor moves it into the appropriate ready queue. A thread that is sleeping for a specified time can also be explicitly awakened by another thread.

A thread that is sleeping *forever* is awakened when it has more work to do. For example, when a thread that is running on a CPU virtual processor needs to access a disk, it issues an I/O request, places itself in a sleep queue for the CPU virtual processor, and yields. When the I/O thread notifies the CPU virtual processor that the I/O is complete, the CPU virtual processor schedules the original thread to continue processing by moving it from the sleep queue to a ready queue. [Figure 5-7](#) illustrates how the database server threads are queued to perform database I/O.

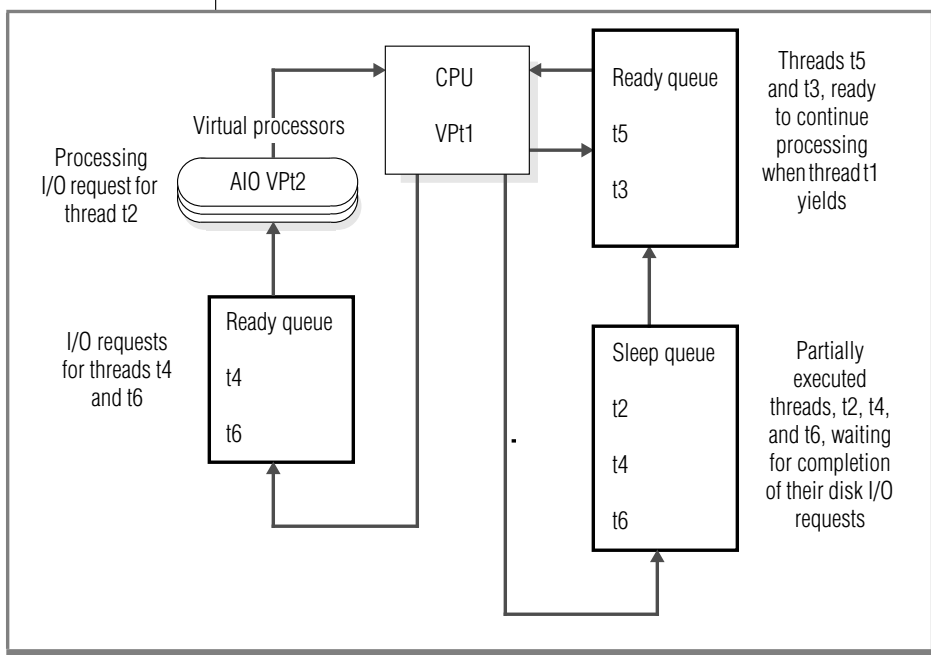


Figure 5-7
How Database Server Threads Are Queued to Perform Database I/O

Wait Queues

Wait queues hold threads that need to wait for a particular event before they can continue to run. For example, wait queues coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch but finds that the latch is held by another user, the thread that was denied access puts itself in the logical-log wait queue. When the thread that owns the lock is ready to release the latch, it checks for waiting threads, and, if threads are waiting, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mutually exclusive*), also referred to as a *latch*, is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources. Mutexes are similar to semaphores, which some operating systems use to regulate access to shared data by multiple *processes*. However, mutexes permit a greater degree of parallelism than semaphores.

A mutex is a variable that is associated with a shared resource such as a buffer. A thread must acquire the mutex for a resource before it can access the resource. Other threads are excluded from accessing the resource until the owner releases it. A thread acquires a mutex, once a mutex becomes available, by setting it to an in-use state. The synchronization that mutexes provide ensures that only one thread at a time writes to an area of shared memory.

For information on monitoring mutexes, refer to [“Monitoring the Shared-Memory Profile and Latches” on page 8-15](#).

Virtual-Processor Classes

A virtual processor of a given class can run only threads of that class. This section describes the types of threads, or the types of processing, that each class of virtual processor performs. It also explains how to determine the number of virtual processors that you need to run for each class.

CPU Virtual Processors

The CPU virtual processor runs all session threads (the threads that process requests from SQL client applications) and some internal threads. Internal threads perform services that are internal to the database server. For example, a thread that listens for connection requests from client applications is an internal thread.

Determining the Number of CPU Virtual Processors Needed

The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You should not allocate more CPU virtual processors than the number of hardware processors in the computer.

To evaluate the performance of the CPU virtual processors while the database server is running, repeat the following command at regular intervals over a set period of time:

```
onstat -g glo
```

If the accumulated *usercpu* and *syscpu* times, taken together, approach 100 percent of the actual elapsed time for the period of the test, add another CPU virtual processor if you have a CPU available to run it.

For information on deciding how many CPU virtual processors you need, see [“Running Poll Threads on CPU or Network Virtual Processors” on page 5-32.](#)

The VPCLASS parameter allows you to specify all of the following information:

- The number of virtual processors to start initially for a class
- The maximum number of virtual processors to run for the class
- Processor affinity for CPU class virtual processors
- Disabling of priority aging, if applicable



Important: *It is recommended that you use the VPCLASS parameter instead of the AFF_SPROC, AFFNPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS parameters. The VPCLASS parameter cannot be used in combination with these parameters. If the ONCONFIG file contains a NUMCPUVPS parameter, for example, you receive an error message if the file also contains a VPCLASS cpu parameter. For information about the VPCLASS configuration parameter, refer to the “Administrator’s Reference.”*

In addition to considering the number of CPUs in the computer and the number of users who will connect to the database server, also consider the fact that user-defined routines and DataBlade modules, which are collections of user-defined routines, run on either CPU virtual processors or user-defined virtual processors. For information on how you assign a user-defined routine to a virtual processor, refer to [“Assigning a UDR to a User-Defined Virtual-Processor Class” on page 5-25.](#)

Running on a Multiprocessor Computer

If you are running multiple CPU virtual processors on a multiprocessor computer, set the MULTIPROCESSOR parameter in the ONCONFIG file to 1. When you set MULTIPROCESSOR to 1, the database server performs locking in a manner that is appropriate for a multiprocessor computer. For information on setting multiprocessor mode, refer to the chapter on configuration parameters in the *Administrator's Reference*.

Running on a Single-Processor Computer

If you are running the database server on a single-processor computer, set the MULTIPROCESSOR configuration parameter to 0. To run the database server with only one CPU virtual processor, set the SINGLE_CPU_VP parameter to 1.

Setting MULTIPROCESSOR to 0 enables the database server to bypass the locking that is required for multiple processes on a multiprocessor computer. For information on the MULTIPROCESSOR configuration parameter, refer to the *Administrator's Reference*.

Setting SINGLE_CPU_VP to 1 allows the database server to bypass some of the mutex calls that it normally makes when it runs multiple CPU virtual processors. For information on setting the SINGLE_CPU_VP parameter, refer to the *Administrator's Reference*.



Important: Setting VPCLASS num to 1 and SINGLE_CPU_VP to 0 does not reduce the number of mutex calls, even though the database server starts only one CPU virtual processor. You must set SINGLE_CPU_VP to 1 to reduce the amount of latching that is performed when you run a single CPU virtual processor.

Setting the SINGLE_CPU_VP parameter to 1 imposes two important restrictions on the database server, as follows:

- Only one CPU virtual processor is allowed.
You cannot add CPU virtual processors while the database server is in online mode.
- No user-defined classes are allowed. (However, users can still define routines that run directly on the CPU VP.)

For more information, see [“Adding Virtual Processors in Online Mode” on page 6-7](#).

Adding and Dropping CPU Virtual Processors in Online Mode

You can add or drop CPU class virtual processors while the database server is online. For instructions on how to do this, see [“Adding Virtual Processors in Online Mode” on page 6-7](#) and [“Dropping CPU and User-Defined Virtual Processors” on page 6-8](#).

Preventing Priority Aging

Some operating systems lower the priority of long-running processes as they accumulate processing time. This feature of the operating system is called *priority aging*. Priority aging can cause the performance of database server processes to decline over time. In some cases, however, the operating system allows you to disable this feature and keep long-running processes running at a high priority.

To determine if priority aging is available on your computer, check the machine notes file described in [“Additional Documentation” on page 13](#) of the Introduction.

If your operating system allows you to disable priority aging, You can disable it by specifying `noage` for the priority entry in the VPCLASS configuration parameter. For more information, refer to the *Administrator's Reference*.

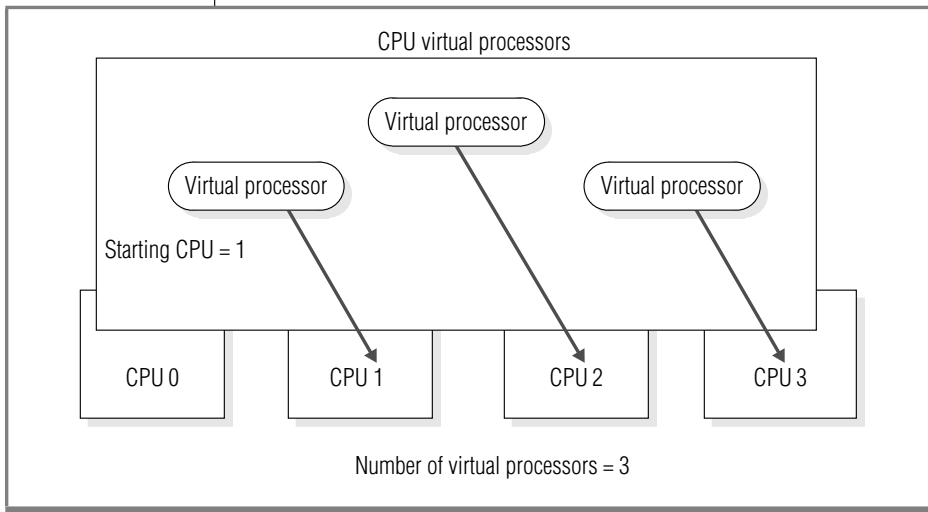
Using Processor Affinity

The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*. Your database server distribution includes a machine notes file that contains information on whether your database server version supports this feature. When you assign a CPU virtual processor to a specific CPU, the virtual processor runs only on that CPU, but other processes also can run on that CPU.

Use the VPCLASS configuration parameter with the *aff* option to implement processor affinity on multiprocessor computers that support it.

Figure 5-8 illustrates the concept of processor affinity.

Figure 5-8
Processor Affinity



UNIX

To see if processor affinity is supported on your UNIX platform, refer to the machine notes file. ♦

Setting Processor Affinity with the VPCLASS Parameter

To set processor affinity with the VPCLASS parameter, specify the range of CPUs to which you are assigning the virtual processors. The database server assigns CPU virtual processors to CPUs in round-robin fashion, starting with the first processor number that *aff* specifies. If you specify more CPU virtual processors than there are processors, the database server starts over again at the beginning. In the following example, the affinity entry assigns a virtual processor to CPUs 1, 2, and 3. Three virtual processors (specified in *num*) are available:

```
VPCLASS CPU,num=3,aff=1-3
```

In the following example, the database server assigns two virtual processors to CPUs 5 and 6, and one virtual processor to CPU 7:

```
VPCLASS CPU,num=3,aff=5-7
```

For more information, see VPCLASS in the chapter on configuration parameters in the *Administrator's Reference*.

User-Defined Classes of Virtual Processors

You can define special classes of virtual processors to run user-defined routines or to run a DataBlade module. User-defined routines are typically written to support user-defined data types. If you do not want a user-defined routine to run in the CPU class, which is the default, you can assign it to a user-defined class of virtual processors (VPs). User-defined classes of virtual processors are also called *extension virtual processors*.

This section provides the following information about user-defined virtual processors:

- When to execute a C-language UDR in a user-defined VP instead of in the CPU VP
- How to assign a C-language UDR to a particular user-defined VP class
- How to add and drop user-defined VPs when the database server is in online mode

Determining the Number of User-Defined Virtual Processors Needed

You can specify as many user-defined virtual processors as your operating system will allow. If you run many UDRs or parallel PDQ queries with UDRs, you should configure more user-defined virtual processors.

Using User-Defined Virtual Processors

User-defined classes of virtual processors protects the database server from *ill-behaved* user-defined routines. An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

A well-behaved C-language UDR has none of these characteristics. Execute only well-behaved C-language UDRs in a CPU VP.



Warning: Execution of an ill-behaved routine in a CPU VP can cause serious interference with the operation of the database server, possibly causing it to fail or behave erratically. In addition, the routine itself might not produce correct results.

To ensure safe execution, assign any ill-behaved user-defined routines to a user-defined class of virtual processors. User-defined VPs remove the following programming restrictions on the CPU VP class:

- The need to yield the processor regularly
- The need to eliminate blocking I/O calls

Functions that run in a user-defined virtual-processor class need not yield the processor, and they might issue direct file-system calls that block further processing by the virtual processor until the I/O is complete.

The normal processing of user queries is not affected by ill-behaved traits of a C-language UDR because these UDRs do not execute in CPU virtual processors. For a more detailed discussion of ill-behaved routines, refer to the *IBM Informix DataBlade API Programmer's Guide*.

Specifying a User-Defined Virtual Processor

The VPCLASS parameter with the *vpclass* option defines a user-defined VP class. You also can specify a nonyielding user-defined virtual processor. For more information, see [“Specifying VPCLASS” on page 6-5](#) and the configuration parameters chapter of the *Administrator's Reference*.

Assigning a UDR to a User-Defined Virtual-Processor Class

The SQL CREATE FUNCTION statement registers a user-defined routine. For example, the following CREATE FUNCTION statement registers the user-defined routine, **GreaterThanEqual()**, and specifies that calls to this routine should be executed by the user-defined VP class named UDR:

```
CREATE FUNCTION GreaterThanEqual (ScottishName, ScottishName)
  RETURNS boolean
  WITH (CLASS = UDR)
  EXTERNAL NAME '/usr/lib/objects/udrs.so'
  LANGUAGE C
```

To execute this function, the ONCONFIG file must include a VPCLASS parameter that defines the UDR class. If not, calls to the **GreaterThanEqual** function will fail.



Tip: The `CLASS` routine modifier can specify any name for the VP class. This class name need not exist when you register the UDR. However, when you try to run a UDR that specifies a user-defined VP class for its execution, this class must exist and have virtual processors assigned to it.

To configure the UDR class, include a line similar to the following one in the `ONCONFIG` file. This line configures the UDR class with two virtual processors and with no priority aging.

```
VPCLASS   UDR,num=2,noage
```

The preceding line defines the UDR VP class as a yielding VP class; that is, this VP class allows the C-language UDR to yield to other threads that need access to the UDR VP class. For more information on how to use the `VPCLASS` configuration parameter, see the *Administrator's Reference*.

For more information on the `CREATE FUNCTION` statement, refer to the *IBM Informix Guide to SQL: Syntax*.

Adding and Dropping User-Defined Virtual Processors in Online Mode

You can add or drop virtual processors in a user-defined class while the database server is online. For instructions on how to do this, refer to [“Adding Virtual Processors in Online Mode” on page 6-7](#) and [“Dropping CPU and User-Defined Virtual Processors” on page 6-8](#).

Java Virtual Processors

Java UDRs and Java applications execute on specialized virtual processors, called *Java virtual processors* (JVPs). A JVP embeds a Java virtual machine (JVM) in its code. A JVP has the same capabilities as a CPU VP in that it can process complete SQL queries.

You can specify as many JVPs as your operating system will allow. If you run many Java UDRs or parallel PDQ queries with Java UDRs, you should configure more JVPs. For more information about UDRs written in Java, refer to *J/Foundation Developer's Guide*.

Use the `VPCLASS` configuration parameter with the `jvp` keyword to configure JVPs. For more information, see the configuration parameters chapter in the *Administrator's Reference*.

Disk I/O Virtual Processors

The following classes of virtual processors perform disk I/O:

- PIO (physical-log I/O)
- LIO (logical-log I/O)
- AIO (asynchronous I/O)
- CPU (kernel-asynchronous I/O)

The PIO class performs all I/O to the physical-log file, and the LIO class performs all I/O to the logical-log files, *unless* those files reside in raw disk space and the database server has implemented KAIO.

On operating systems that do not support KAIO, the database server uses the AIO class of virtual processors to perform database I/O that is not related to physical or logical logging.

The database server uses the CPU class to perform KAIO when it is available on a platform. If the database server implements KAIO, a KAIO thread performs all I/O to raw disk space, including I/O to the physical and logical logs.

UNIX

To find out if your UNIX platform supports KAIO, refer to the machine notes file. ♦

Windows

Windows supports KAIO. ♦

For more information about nonlogging I/O, refer to [“Asynchronous I/O” on page 5-29](#).

I/O Priorities

In general, the database server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the nonlogging I/O queues. Prioritizing ensures that a high-priority log I/O, for example, is never queued behind a write to a temporary file, which has a low priority. The database server prioritizes the different types of disk I/O that it performs, as [Figure 5-9](#) shows.

Figure 5-9
How Database Server Prioritizes Disk I/O

Priority	Type of I/O	VP Class
1st	Logical-log I/O	CPU or LIO
2nd	Physical-log I/O	CPU or PIO
3rd	Database I/O	CPU or AIO
3rd	Page-cleaning I/O	CPU or AIO
3rd	Read-ahead I/O	CPU or AIO

Logical-Log I/O

The LIO class of virtual processors performs I/O to the logical-log files in the following cases:

- KAIO is not implemented.
- The logical-log files are in cooked disk space.

Only when KAIO is implemented and the logical-log files are in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the logical log.

The logical-log files store the data that enables the database server to roll back transactions and recover from system failures. I/O to the logical-log files is the highest priority disk I/O that the database server performs.

If the logical-log files are in a dbspace that *is not* mirrored, the database server runs only one LIO virtual processor. If the logical-log files are in a dbspace that *is* mirrored, the database server runs two LIO virtual processors. This class of virtual processors has no parameters associated with it.

Physical-Log I/O

The PIO class of virtual processors performs I/O to the physical-log file in the following cases:

- KAIO is not implemented.
- The physical-log file is stored in buffered-file chunks.

Only when KAIO is implemented and the physical-log file is in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the physical log. The physical-log file stores *before-images* of dbspace pages that have changed since the last *checkpoint*. (For more information on checkpoints, refer to “[Checkpoints](#)” on page 15-11.) At the start of recovery, prior to processing transactions from the logical log, the database server uses the physical-log file to restore before-images to dbspace pages that have changed since the last checkpoint. I/O to the physical-log file is the second-highest priority I/O after I/O to the logical-log files.

If the physical-log file is in a dbspace that *is not* mirrored, the database server runs only one PIO virtual processor. If the physical-log file is in a dbspace that *is* mirrored, the database server runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

The database server performs database I/O asynchronously, meaning that I/O is queued and performed independently of the thread that requests the I/O. Performing I/O asynchronously allows the thread that makes the request to continue working while the I/O is being performed.

The database server performs all database I/O asynchronously, using one of the following facilities:

- AIO virtual processors
- KAIO on platforms that support it

Database I/O includes I/O for SQL statements, read-ahead, page cleaning, and checkpoints, as well as other I/O.

Kernel-Asynchronous I/O

The database server uses KAIO when the following conditions exist:

- The computer and operating system support it.
- A performance gain is realized.
- The I/O is to raw disk space.

UNIX

The database server implements KAIO by running a KAIO thread on the CPU virtual processor. The KAIO thread performs I/O by making system calls to the operating system, which performs the I/O independently of the virtual processor. The KAIO thread can produce better performance for disk I/O than the AIO virtual processor can, because it does not require a switch between the CPU and AIO virtual processors.

Informix implements KAIO when it ports the database server to a platform that supports this feature. The database server administrator does not configure KAIO. To see if KAIO is supported on your platform, see the machine notes file. ♦

AIO Virtual Processors

If the platform does not support KAIO or if the I/O is to buffered-file chunks, the database server performs database I/O through the AIO class of virtual processors. All AIO virtual processors service all I/O requests equally within their class.

The database server assigns each disk chunk a queue, sometimes known as a *gfd queue*, based on the filename of the chunk. The database server orders I/O requests within a queue according to an algorithm that minimizes disk-head movement. The AIO virtual processors service queues that have work pending in round-robin fashion.

All other non-chunk I/O is queued in the AIO queue.

Use the VPCLASS parameter with the *aio* keyword to specify the number of AIO virtual processors that the database server starts initially. For information about VPCLASS, refer to the chapter on configuration parameters in the *Administrator's Reference*.

You can start additional AIO virtual processors while the database server is in online mode. For more information, refer to [“Adding Virtual Processors in Online Mode” on page 6-7](#).

You cannot drop AIO virtual processors while the database server is in online mode.

Number of AIO Virtual Processors Needed

The goal in allocating AIO virtual processors is to allocate enough of them so that the lengths of the I/O request queues are kept short; that is, the queues have as few I/O requests in them as possible. When the *gfd* queues are consistently short, it indicates that I/O to the disk devices is being processed as fast as the requests occur. The **onstat -g ioq** command allows you to monitor the length of the *gfd* queues for the AIO virtual processors. For more information, refer to [“Monitoring Virtual Processors” on page 6-9](#).

If the database server implements KAIO on your platform, and all of your dbspaces are composed of raw disk space, one AIO virtual processor might be sufficient.

If the database server implements KAIO, but you are using some buffered files for chunks, allocate two AIO virtual processors per active dbspace that is composed of buffered file chunks. If KAIO is *not* implemented on your platform, allocate two AIO virtual processors for each disk that the database server accesses frequently.

Allocate enough AIO virtual processors to accommodate the peak number of I/O requests. Generally, it is not detrimental to allocate too many AIO virtual processors.

Network Virtual Processors

As explained in [Chapter 3, “Client/Server Communications,”](#) a client can connect to the database server in the following ways:

- Through a network connection
- Through a pipe
- Through shared memory ♦

The network connection can be made by a client on a remote computer or by a client on the local computer mimicking a connection from a remote computer (called a *local-loopback connection*).

Specifying Network Connections

In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the **sqlhosts** file or registry. Each dbservername parameter in **sqlhosts** has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more *poll threads* for each unique **nettype** entry. For a description of the **nettype** field, refer to [“The Connection Type Field” on page 3-34](#).

The NETTYPE configuration parameter provides optional configuration information for an interface/protocol combination. It allows you to allocate more than one poll thread for an interface/protocol combination and also designate the virtual-processor class (CPU or NET) on which the poll threads run. For a complete description of this configuration parameter, refer to the *Administrator's Reference*.

Running Poll Threads on CPU or Network Virtual Processors

Poll threads can run either on CPU virtual processors or, depending on the connection type, on network virtual processors. In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This might not be true, however, on a multiprocessor computer with a large number of remote clients.

The NETTYPE parameter has an optional entry, called `vp class`, that allows you to specify either CPU or NET, for CPU or network virtual-processor classes, respectively.

If you do not specify a virtual processor class for the interface/protocol combination (poll threads) associated with the DBSERVERNAME variable, the class defaults to CPU. The database server assumes that the interface/protocol combination associated with DBSERVERNAME is the primary interface/protocol combination and that it should be the most efficient.

For other interface/protocol combinations, if no `vp class` is specified, the default is NET.

While the database server is in online mode, you cannot drop a CPU virtual processor that is running a poll thread.

Specifying the Number of Networking Virtual Processors

Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by the NET class. If you specify CPU for the `vp class`, you must allocate a sufficient number of CPU virtual processors to run the poll threads. If the database server does not have a CPU virtual processor to run a CPU poll thread, it starts a network virtual processor of the specified class to run it.

For most systems, one poll thread and consequently one virtual processor per network interface/protocol combination is sufficient. For systems with 200 or more network users, running additional network virtual processors might improve throughput. In this case, you need to experiment to determine the optimal number of virtual processors for each interface/protocol combination.

Specifying Listen and Poll Threads for the Client/Server Connection

When you start the database server, the **oninit** process starts an internal thread, called a *listen thread*, for each `dbservername` that you specify with the `DBSERVERNAME` and `DBSERVERALIASES` parameters in the `ONCONFIG` file. To specify a listen port for each of these `dbservername` entries, assign it a unique combination of **hostname** and **service name** entries in `sqlhosts`. For example, the `sqlhosts` file or registry entry shown in [Figure 5-10](#) causes the database server `soc_ol1` to start a listen thread for **port1** on the host, or network address, `myhost`.

Figure 5-10
A Listen Thread for Each Listen Port

dbservername	nettype	hostname	service name
soc_ol1	onsocket	myhost	port1

The listen thread opens the port and requests one of the poll threads for the specified interface/protocol combination to monitor the port for client requests. The poll thread runs either in the CPU virtual processor or in the network virtual processor for the connection that is being used. For information on the number of poll threads, refer to [“Specifying the Number of Networking Virtual Processors” on page 5-33](#).

For information on how to specify whether the poll threads for an interface/protocol combination run in CPU or network virtual processors, refer to [“Running Poll Threads on CPU or Network Virtual Processors” on page 5-32](#) and to the NETTYPE configuration parameter in the *Administrator’s Reference*.

When a poll thread receives a connection request from a client, it passes the request to the listen thread for the port. The listen thread authenticates the user, establishes the connection to the database server, and starts an **sqlxec** thread, the session thread that performs the primary processing for the client. [Figure 5-11](#) illustrates the roles of the listen and poll threads in establishing a connection with a client application.

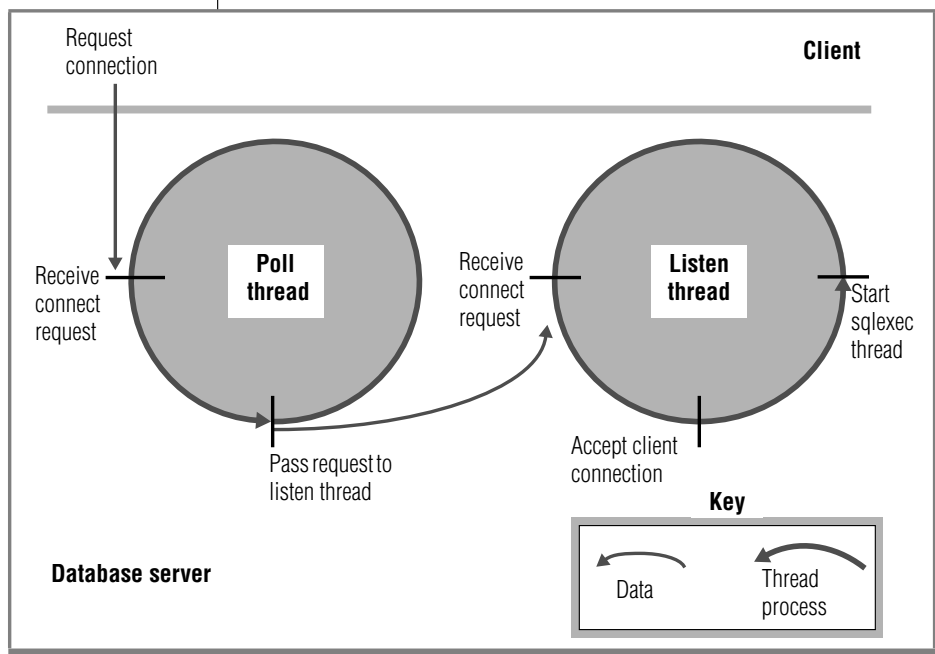


Figure 5-11
The Roles of the Poll
and the Listen
Threads in
Connecting to a
Client

A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlxec** thread. For network connections, the poll thread places the message in a queue in the shared-memory global pool. The poll thread then wakes up the **sqlxec** thread of the client to process the request. Whenever possible, the **sqlxec** thread writes directly back to the client without the help of the poll thread. In general, the poll thread reads data from the client, and the **sqlxec** thread sends data to the client.

UNIX

For a shared-memory connection, the poll thread places the message in the communications portion of shared memory. ♦

Figure 5-12 illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

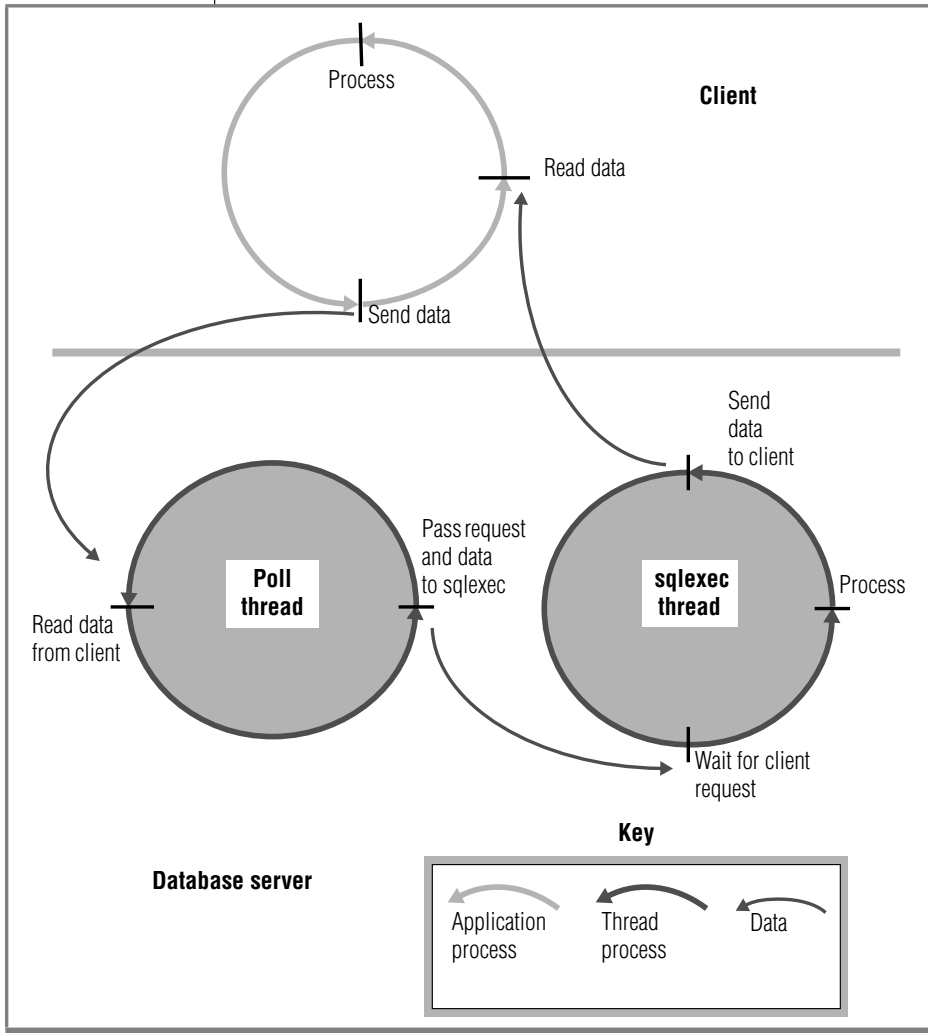


Figure 5-12
The Roles of the Poll
and sqlexec Threads
in Communicating
with the Client
Application

Starting Multiple Listen Threads

If the database server cannot service connection requests satisfactorily for a given interface/protocol combination with a single port and corresponding listen thread, you can improve service for connection requests in the following two ways:

- Add listen threads for additional ports.
- Add another network-interface card.

Adding Listen Threads

As stated previously, the database server starts a listen thread for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES configuration parameters.

To add listen threads for additional ports, you must first use the DBSERVERALIASES parameter to specify dbservernames for each of the ports. For example, the DBSERVERALIASES parameter in [Figure 5-13](#) defines two additional dbservernames, **soc_ol2** and **soc_ol3**, for the database server instance identified as **soc_ol1**.

DBSERVERNAME	soc_ol1
DBSERVERALIASES	soc_ol2, soc_ol3

Figure 5-13
Defining Multiple dbservernames for Multiple Connections of the Same Type

Once you define additional dbservernames for the database server, you must specify an interface/protocol combination and port for each of them in the **sqlhosts** file or registry. Each port is identified by a unique combination of **hostname** and **servicename** entries. For example, the **sqlhosts** entries shown in [Figure 5-14](#) cause the database server to start three listen threads for the **onsoctcp** interface/protocol combination, one for each of the ports defined.

Figure 5-14
sqlhosts Entries to Listen to Multiple Ports for a Single Interface/Protocol Combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1
soc_ol2	onsoctcp	myhost	port2
soc_ol3	onsoctcp	myhost	port3

If you include a NETTYPE parameter for an interface/protocol combination, it applies to all the connections for that interface/protocol combination. In other words, if a NETTYPE parameter exists for **onsoctcp** in [Figure 5-14](#), it applies to all of the connections shown. In this example, the database server runs one *poll* thread for the **onsoctcp** interface/protocol combination unless the NETTYPE parameter specifies more. For more information about entries in the **sqlhosts** file or registry, refer to [“Connectivity Files” on page 3-13](#).

Adding a Network-Interface Card

If the network-interface card for the host computer cannot service connection requests satisfactorily, or if you want to connect the database server to more than one network, you can add a network-interface card.

To support multiple network-interface cards, you must assign each card a unique **hostname** (network address) in **sqlhosts**. For example, using the same dbservernames shown in [Figure 5-13](#), the **sqlhosts** file or registry entries shown in [Figure 5-15 on page 5-37](#) cause the database server to start three listen threads for the same interface/protocol combination (as did the entries in [Figure 5-14](#)). In this case, however, two of the threads are listening to ports on one interface card (**myhost1**), and the third thread is listening to a port on the second interface card (**myhost2**).

Figure 5-15
Example of sqlhosts Entries to Support Two Network-Interface Cards for the onsoctcp Interface/Protocol Combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost1	port1
soc_ol2	onsoctcp	myhost1	port2
soc_ol3	onsoctcp	myhost2	port1

Communications Support Module Virtual Processor

The communications support module (CSM) class of virtual processors performs communications support service and communications support module functions.

The database server starts the same number of CSM virtual processors as the number of CPU virtual processors that it starts.

For more information on the communications support service, refer to [Chapter 3, “Client/Server Communications.”](#)

Optical Virtual Processor

The optical class (OPT) of virtual processors is used only with the Optical Subsystem. The Optical Subsystem starts one virtual processor in the optical class if the STAGEBLOB configuration parameter is present. For more information on the Optical Subsystem, see the *IBM Informix Optical Subsystem Guide*.

Audit Virtual Processor

The database server starts one virtual processor in the audit class (ADT) when you turn on audit mode by setting the ADTMODE parameter in the ONCONFIG file to 1. For more information about database server auditing, refer to your *IBM Informix Trusted Facility Guide*.

Miscellaneous Virtual Processor

The miscellaneous virtual processor services requests for system calls that might require a very large stack, such as fetching information about the current user or the host-system name. Only one thread runs on this virtual processor; it executes with a stack of 128 kilobytes.

Managing Virtual Processors

In This Chapter	6-3
Setting Virtual-Processor Configuration Parameters	6-3
Setting Virtual-Processor Parameters with a Text Editor	6-4
Specifying VPCLASS	6-5
Disabling Priority Aging	6-5
Setting Virtual-Processor Parameters with ISA	6-5
Setting Virtual-Processor Parameters with ON-Monitor	6-6
Starting and Stopping Virtual Processors	6-6
Adding Virtual Processors in Online Mode	6-7
Adding Virtual Processors in Online Mode with onmode	6-7
Adding Virtual Processors in Online Mode with ON-Monitor	6-8
Adding Network Virtual Processors	6-8
Dropping CPU and User-Defined Virtual Processors	6-8
Monitoring Virtual Processors	6-9
Monitoring Virtual Processors with Command-Line Utilities	6-9
onstat -g ath	6-10
onstat -g glo	6-10
onstat -g ioq	6-11
onstat -g rea	6-11
Monitoring Virtual Processors with SMI Tables	6-12

In This Chapter

This chapter describes how to set the configuration parameters that affect database server virtual processors, and how to start and stop virtual processors.

For descriptions of the virtual-processor classes and for advice on how many virtual processors you should specify for each class, refer to [Chapter 5, “Virtual Processors and Threads.”](#)

Setting Virtual-Processor Configuration Parameters

As user **root** or **informix**, use the following tools to set the configuration parameters for the database server virtual processors:

- A text editor
- IBM Informix Server Administrator (ISA)
- ON-Monitor ♦

To implement any changes that you make to configuration parameters, you must shut down and restart the database server. For more information, refer to [“Reinitializing Shared Memory” on page 8-12.](#)

Setting Virtual-Processor Parameters with a Text Editor

You can use a text editor program to set ONCONFIG parameters at any time. Use the editor to locate the parameter that you want to change, enter the new value, and rewrite the file to disk.

[Figure 6-1](#) lists the ONCONFIG parameters that are used to configure virtual processors. For more information on how these parameters affect virtual processors, refer to [“Virtual-Processor Classes” on page 5-19](#).

Figure 6-1
ONCONFIG Parameters for Configuring Virtual Processors

Parameter	Subparameters	Purpose
MULTIPROCESSOR		Specifies that you are running on a multiprocessor computer
NETTYPE		Specifies parameters for network protocol threads (and virtual processors)
SINGLE_CPU_VP		Specifies that you are running a single CPU virtual processor
VPCLASS		Specifies the following items:
	adm adt cpu kio lio msc opt lio shm soc str tli	■ A class of virtual processors
	num=num_VPs	■ The number of virtual processors of the specified class that the database server should start
	max=max_VPs	■ The maximum number of virtual processors allowed for a class
	noage	■ Disabling of priority aging
	aff=processor_number aff=start_range - end_range	■ Assignment to CPUs if processor affinity is available

(1 of 2)

Parameter	Subparameters	Purpose
	<code>user_defined</code>	■ User-defined virtual processor
	<code>jvp</code>	■ Java virtual processor (JVP)
	<code>noyield</code>	■ Non-yielding virtual processor

(2 of 2)

Specifying VPCLASS

It is recommended that you use the VPCLASS parameter instead of the NUMCPUVPS, NUMAIOVPS, NOAGE, AFF_SPROC, and AFF_NPROCS parameters. You can specify a VPCLASS name of up to 128 characters. The VPCLASS name must begin with a letter or underscore, and must contain letters, digits, underscores, or \$ characters.

You specify user-defined classes of virtual processors and Java virtual processors in the VPCLASS configuration parameter. For more information on the VPCLASS configuration parameter, including the defaults and value ranges, see the *Administrator's Reference*, [“User-Defined Classes of Virtual Processors” on page 5-24](#), and [“Java Virtual Processors” on page 5-26](#).

Disabling Priority Aging

Use the VPCLASS parameter with the *noage* option to disable process priority aging on platforms that allow this feature.

For recommended values for these database server parameters on UNIX, refer to your machine notes file.

Setting Virtual-Processor Parameters with ISA

You can use ISA to display information about virtual processor classes, and monitor, add, or remove virtual-processor classes. For more information, see the ISA online help.

UNIX

Setting Virtual-Processor Parameters with ON-Monitor

To set the virtual-processor configuration parameters with ON-Monitor, select the **Parameters**→**perFormance** option.

To specify network virtual processors, enter the number of virtual processors and then enter one of the following interface/protocol combinations: **ipcshm**, **ipcstr**, **tlitcp**, **tlispx**, or **soctcp**.

Starting and Stopping Virtual Processors

When you start the database server, the **oninit** utility starts the number and types of virtual processors that you have specified directly and indirectly. You configure virtual processors primarily through ONCONFIG parameters and, for network virtual processors, through parameters in the **sqlhosts** file or registry. For descriptions of the virtual-processor classes, refer to [“Virtual-Processor Classes” on page 5-19](#).

The database server allows you to start a maximum of 1000 virtual processors.

Once the database server is in online mode, you can start additional virtual processors to improve performance, if necessary. For more information, refer to [“Adding Virtual Processors in Online Mode”](#) below.

While the database server is in online mode, you can drop virtual processors of the CPU and user-defined classes. For more information, refer to [“Dropping CPU and User-Defined Virtual Processors” on page 6-8](#).

To terminate the database server and thereby terminate all virtual processors, use the **onmode -k** command. For more information on using **onmode -k**, refer to the utilities chapter of the *Administrator's Reference*.

Adding Virtual Processors in Online Mode

While the database server is in online mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, STR, TLI, SOC, JVP, and user-defined. The database server automatically starts one virtual processor each in the LIO and PIO classes unless mirroring is used, in which case it starts two.

You can start these additional virtual processors with one of the following methods:

- The **-p** option of the **onmode** utility
- ISA

You can also start additional virtual processors for user-defined classes to run user-defined routines. For more information about user-defined virtual processors, refer to [“Assigning a UDR to a User-Defined Virtual-Processor Class” on page 5-25](#).

Adding Virtual Processors in Online Mode with onmode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in online mode. Specify the number of virtual processors that you want to add with a positive number. As an option, you can precede the number of virtual processors with a plus sign (+). Following the number, specify the virtual processor class in lowercase letters. For example, either of the following commands starts four additional virtual processors in the AIO class:

```
onmode -p 4 aio
```

```
onmode -p +4 aio
```

The **onmode** utility starts the additional virtual processors immediately.

You can add virtual processors to only one class at a time. To add virtual processors for another class, you must run **onmode** again.

UNIX

Adding Virtual Processors in Online Mode with ON-Monitor

To use ON-Monitor to add virtual processors while the database server is online, select the **Modes→Add-Proc** option. You can add virtual processors in the following classes: CPU, AIO, LIO, PIO, and NET.

To specify network virtual processors, first enter the number of virtual processors and then enter one of the following interface/protocol combinations: **ipcshm**, **ipcstr**, **tlitcp**, **tlispx**, or **soctcp**.

You cannot use ON-Monitor to start additional virtual processors for a user-defined class. For more information, refer to [“Adding Virtual Processors in Online Mode with onmode.”](#)

Adding Network Virtual Processors

When you add network virtual processors, you are adding poll threads, each of which requires its own virtual processor to run. If you attempt to add poll threads for a protocol while the database server is in online mode, and you have specified in the NETTYPE parameter that the poll threads run in the CPU class, the database server does not start the new poll threads if no CPU virtual processors are available to run them.

Dropping CPU and User-Defined Virtual Processors

While the database server is in online mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU and user-defined classes.

To drop CPU virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the CPU class in lowercase letters. For example, the following command drops two CPU virtual processors:

```
% onmode -p -2 cpu
```

If you attempt to drop a CPU virtual processor that is running a poll thread, you receive the following message:

```
onmode: failed when trying to change the number of cpu  
virtual processor by -number.
```

For more information, refer to [“Running Poll Threads on CPU or Network Virtual Processors”](#) on page 5-32.

To drop user-defined virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the user-defined class in lowercase letters. For example, the following command drops two virtual processors of the class *usr*:

```
onmode -p -2 usr
```

Windows

In Windows, you can have only one user-defined virtual processor class at a time. Omit the *number* parameter in the **onmode -p *vpclass*** command. ♦

For information on how to create a user-defined class of virtual processors and assign user-defined routines to it, refer to [“User-Defined Classes of Virtual Processors”](#) on page 5-24.

Monitoring Virtual Processors

Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity. For more information on these **onstat -g** options, see the chapter on the effect of configuration on CPU utilization in the *Performance Guide*.

Monitoring Virtual Processors with Command-Line Utilities

You can use the following **onstat -g** options to monitor virtual processors:

- **ath**
- **glo**
- **ioq**
- **rea**

onstat -g ath

The **onstat -g ath** command displays information on system threads and the virtual-processor classes.

onstat -g glo

Use the **onstat -g glo** command to display information about each virtual processor that is currently running, as well as cumulative statistics for each virtual processor class. [Figure 6-2](#) shows an example of the output from this option.

```
MT global info:
sessions threads vps lngspins
1 15 8 0
```

```
Virtual processor summary:
class vps usercpu syscpu total
cpu 3 479.77 190.42 670.18
aio 1 0.83 0.23 1.07
pio 1 0.42 0.10 0.52
lio 1 0.27 0.22 0.48
soc 0 0.00 0.00 0.00
tli 0 0.00 0.00 0.00
shm 0 0.00 0.00 0.00
adm 1 0.10 0.45 0.55
opt 0 0.00 0.00 0.00
msc 1 0.28 0.52 0.80
adt 0 0.00 0.00 0.00
total 8 481.67 191.93 673.60
```

```
Individual virtual processors:
vp pid class usercpu syscpu total
1 1776 cpu 165.18 40.50 205.68
2 1777 adm 0.10 0.45 0.55
3 1778 cpu 157.83 98.68 256.52
4 1779 cpu 156.75 51.23 207.98
5 1780 lio 0.27 0.22 0.48
6 1781 pio 0.42 0.10 0.52
7 1782 aio 0.83 0.23 1.07
8 1783 msc 0.28 0.52 0.80
tot 481.67 191.93 673.60
```

Figure 6-2
onstat -g glo Output

onstat -g ioq

Use the **onstat -g ioq** option to determine whether you need to allocate additional AIO virtual processors. The command **onstat -g ioq** displays the length of the I/O queues under the column **len**, as [Figure 6-3](#) shows.

If the length of the I/O queue is growing, I/O requests are accumulating faster than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.

```
onstat -g ioq
```

AIO I/O queues:

q	name/id	len	maxlen	totalops	dskread	dskwrite	dskcopy
adt	0	0	0	0	0	0	0
msc	0	0	1	12	0	0	0
aio	0	0	4	89	68	0	0
pio	0	0	1	1	0	1	0
lio	0	0	1	17	0	17	0
kio	0	0	0	0	0	0	0
gfd	3	0	3	254	242	12	0
gfd	4	0	17	614	261	353	0

Figure 6-3
*onstat -g ioq and
onstat -d Output*

onstat -g rea

Use the **onstat -g rea** option to monitor the number of threads in the ready queue. If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might have to add more virtual processors to your configuration. [Figure 6-4](#) shows **onstat -g rea** output.

Ready threads:

tid	tcb	rstcb	prty	status	vp-class	name
6	536a38	406464	4	ready	3cpu	main_loop()
28	60cfe8	40a124	4	ready	1cpu	onmode_mon
33	672a20	409dc4	2	ready	3cpu	sqlxec

Figure 6-4
onstat -g rea Output

Monitoring Virtual Processors with SMI Tables

Query the **sysvppprof** table to obtain information on the virtual processors that are currently running. This table contains the following columns.

Column	Description
vpid	Virtual-processor ID number
class	Virtual-processor class
usercpu	Minutes of user CPU consumed
syscpu	Minutes of system CPU consumed

Shared Memory

In This Chapter	7-5
Shared Memory	7-5
Shared-Memory Use	7-6
Shared-Memory Allocation.	7-7
Shared-Memory Size	7-9
Action to Take If SHMTOTAL Is Exceeded	7-10
Processes That Attach to Shared Memory	7-10
How a Client Attaches to the Communications Portion	7-11
How Utilities Attach to Shared Memory	7-11
How Virtual Processors Attach to Shared Memory	7-12
Obtaining Key Values for Shared-Memory Segments	7-13
Specifying Where to Attach the First Shared-Memory Segment	7-13
Attaching Additional Shared-Memory Segments	7-14
Defining the Shared-Memory Lower-Boundary Address	7-14
Resident Shared-Memory Segments	7-16
Resident Portion of Shared Memory	7-16
Shared-Memory Header.	7-17
Shared-Memory Buffer Pool	7-17
Buffer Overflow to the Virtual Portion	7-18
Buffer Size	7-18
Logical-Log Buffer.	7-19
Physical-Log Buffer	7-20
High-Availability Data-Replication Buffer	7-21
Lock Table	7-21

Virtual Portion of Shared Memory	7-22
Management of the Virtual Portion of Shared Memory	7-23
Size of the Virtual Portion of Shared Memory	7-23
Components of the Virtual Portion of Shared Memory	7-24
Shared-Memory Internal Tables	7-24
Big Buffers	7-28
Session Data	7-28
Thread Data.	7-28
Data-Distribution Cache	7-30
Dictionary Cache	7-30
SQL Statement Cache	7-31
Sorting Memory	7-31
SPL Routine and the UDR Cache	7-31
Global Pool	7-32
Communications Portion of Shared Memory	7-32
Virtual-Extension Portion of Shared Memory	7-32
Concurrency Control	7-33
Shared-Memory Mutexes	7-33
Shared-Memory Buffer Locks	7-33
Types of Buffer Locks	7-34
Database Server Thread Access to Shared Buffers	7-34
LRU Queues	7-34
Components of LRU Queue	7-35
Pages in Least-Recently Used Order	7-35
LRU Queues and Buffer-Pool Management	7-36
Number of LRU Queues to Configure.	7-36
Number of Cleaners to Allocate	7-37
Number of Pages Added to the MLRU Queues	7-38
End of MLRU Cleaning.	7-38
Configuring the Database Server to Read Ahead	7-39
Database Server Thread Access to Buffer Pages	7-40
Flushing Data to Disk	7-40
Flushing Buffer-Pool Buffers	7-41
Flushing Before-Images First	7-41
Flushing the Physical-Log Buffer	7-42
Synchronizing Buffer Flushing	7-42
Describing Flushing Activity	7-43

Foreground Write	7-43
LRU Write	7-44
Chunk Write	7-44
Flushing the Logical-Log Buffer.	7-45
After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging	7-45
When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates	7-46
When a Checkpoint Occurs	7-46
When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File.	7-46
Buffering Large-Object Data	7-46
Writing Simple Large Objects	7-47
Blobpages and Shared Memory	7-47
Creation of Simple Large Objects	7-47
Creation of Blobpage Buffers	7-48
Accessing Smart Large Objects	7-49
Memory Use on 64-Bit Platforms.	7-50

In This Chapter

This chapter describes the content of database server shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory. For information on how to change the database server configuration parameters that determine shared memory allocations, refer to [Chapter 8, “Managing Shared Memory.”](#)

Shared Memory

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory. The database server uses shared memory for the following purposes:

- To reduce memory usage and disk I/O
- To perform high-speed communication between processes

Shared memory enables the database server to reduce overall memory usage because the participating processes, in this case, virtual processors, do not need to maintain private copies of the data that is in shared memory.

Shared memory reduces disk I/O, because buffers, which are managed as a common pool, are flushed on a database server-wide basis instead of a per-process basis. Furthermore, a virtual processor can often avoid reading data from disk because the data is already in shared memory as a result of an earlier read operation. The reduction in disk I/O reduces execution time.

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

Shared-Memory Use

The database server uses shared memory for the following purposes:

- To enable virtual processors and utilities to share data
- To provide a fast communications channel for local client applications that use IPC communication

Figure 7-1 illustrates the shared-memory scheme.

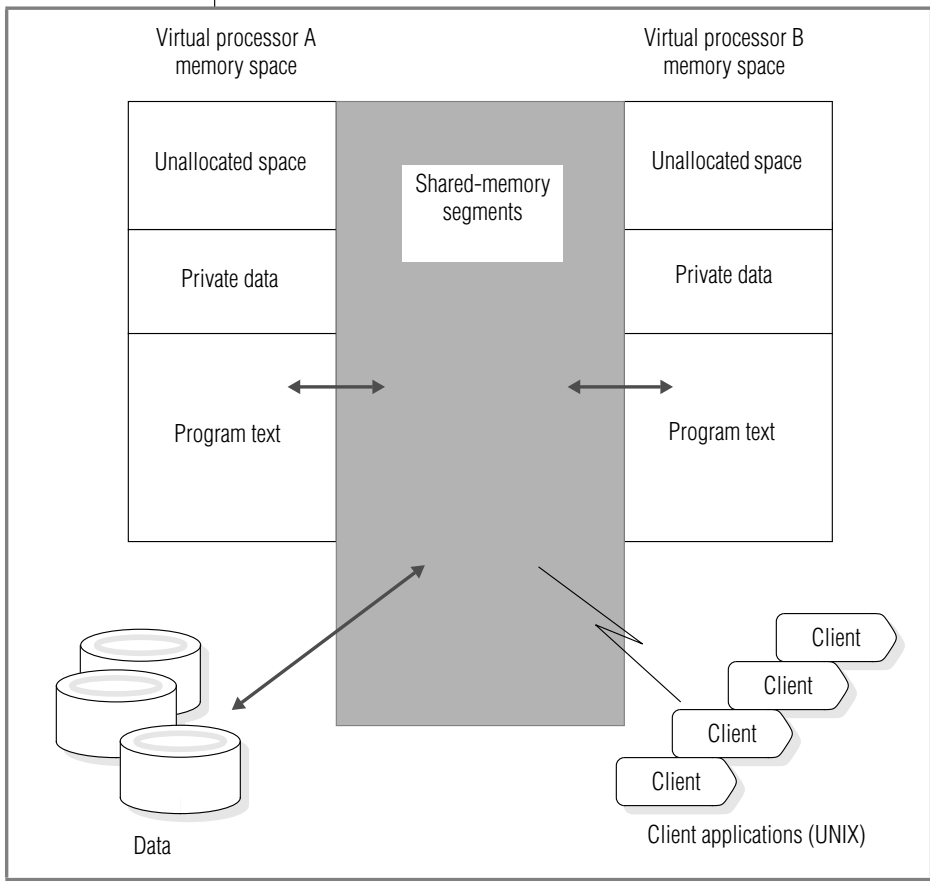


Figure 7-1
How the Database Server Uses Shared Memory

UNIX

Shared-Memory Allocation

The database server creates the following portions of shared memory:

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

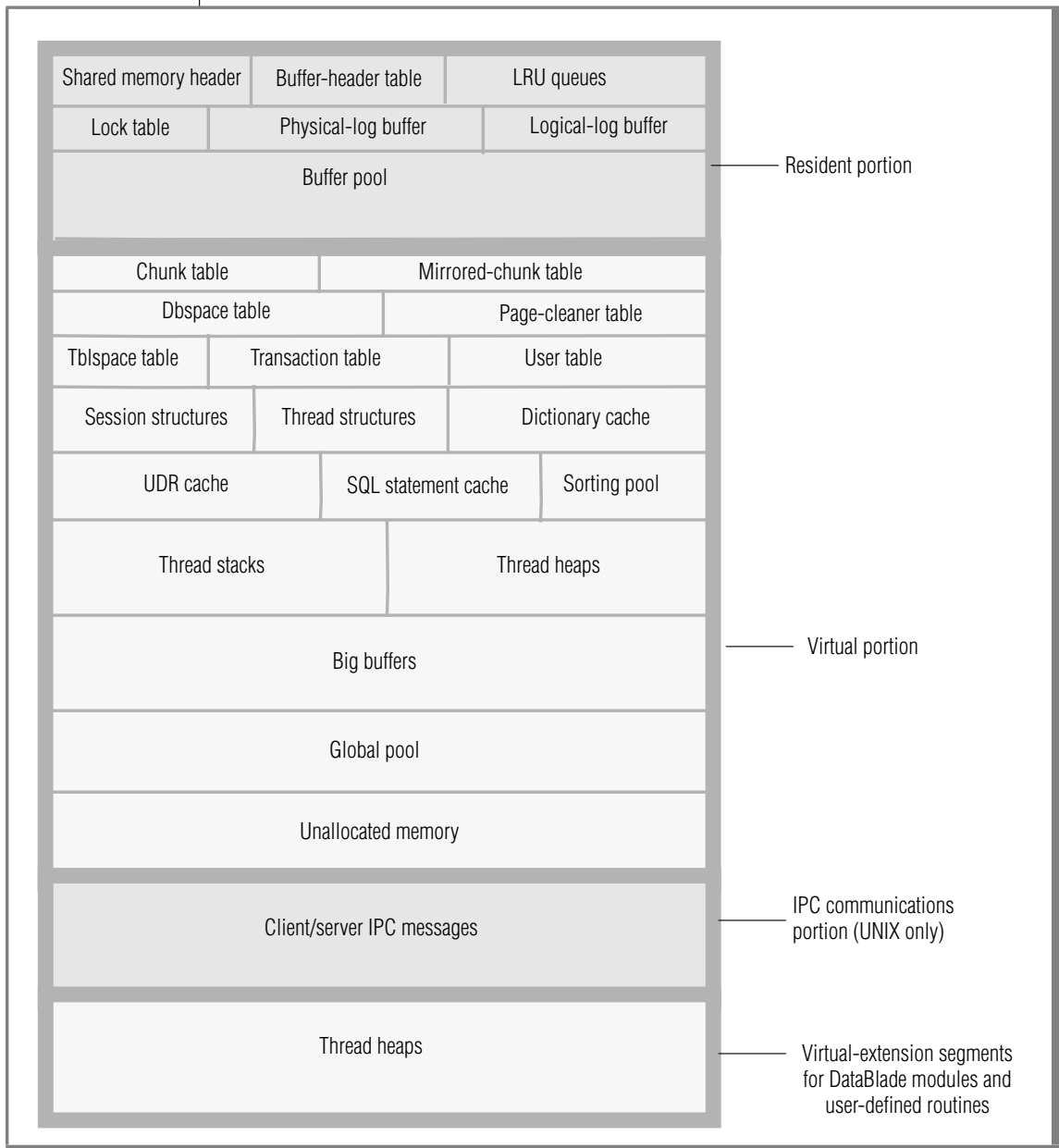
If the **sqlhosts** file specifies shared-memory communications, the database server allocates memory for the communications portion. ♦

- The *virtual-extension* portion

The database server adds operating-system segments as needed to the virtual and virtual-extension portions of shared memory. For more information on shared-memory settings for your platform, see the machine notes. [Figure 7-2 on page 7-8](#) shows the contents of each portion of shared memory.

All database server virtual processors have access to the same shared-memory segments. Each virtual processor manages its work by maintaining its own set of pointers to shared-memory resources such as buffers, locks, and latches. Virtual processors attach to shared memory when you take the database server from offline mode to quiescent mode or from offline mode directly to online mode. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple threads.

Figure 7-2
Contents of Database Server Shared Memory



Shared-Memory Size

Each portion of the database server shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks that are 4 kilobytes in size and managed by a bitmap.

The header-line output by the **onstat** utility contains the size of the database server shared memory, expressed in kilobytes. For information on how to use **onstat**, refer to the utilities chapter in the *Administrator's Reference*. You can also use `onstat -g seg` to monitor how much memory the database server allocates for each portion of shared memory.

You can set the SHMTOTAL parameter in the ONCONFIG file to limit the amount of memory overhead that the database server can place on your computer or node. The SHMTOTAL parameter specifies the total amount of shared memory that the database server can use for all memory allocations. However, certain operations might fail if the database server needs more memory than the amount set in SHMTOTAL. If this condition occurs, the database server displays the following message in the message log:

```
size of resident + virtual segments x + y > z
total allowed by configuration parameter SHMTOTAL
```

In addition, the database server returns an error message to the application that initiated the offending operation. For example, if the database server needs more memory than you specify in SHMTOTAL while it tries to perform an operation such as an index build or a hash join, it returns an error message to the application that is similar to one of the following:

```
-567      Cannot write sorted rows.
-116      ISAM error: cannot allocate memory.
```

After the database server sends these messages, it rolls back any partial results performed by the offending query.

Internal operations, such as page-cleaner or checkpoint activity, can also cause the database server to exceed the SHMTOTAL ceiling. When this situation occurs, the database server sends a message to the message log. For example, suppose that the database server attempts and fails to allocate additional memory for page-cleaner activity. As a consequence, the database server sends a message to the message log that is similar to the following:

```
17:19:13 Assert Failed: WARNING! No memory available for page cleaners
17:19:13 Who: Thread(11, flush_sub(0), 9a8444, 1)
17:19:13 Results: Database server may be unable to complete a checkpoint
17:19:13 Action: Make more virtual memory available to database server
17:19:13 See Also: /tmp/af.c4
```

After the database server informs you about the failure to allocate additional memory, it rolls back the transactions that caused it to exceed the SHMTOTAL limit. Immediately after the rollback, operations no longer fail from lack of memory, and the database server continues to process transactions as usual.

Action to Take If SHMTOTAL Is Exceeded

When the database server needs more memory than SHMTOTAL allows, a transient condition occurs, perhaps caused by a burst of activity that exceeds the normal processing load. Only the operation that caused the database server to run out of memory temporarily should fail. Other operations continue to be processed in a normal fashion.

If messages indicate on a regular basis that the database server needs more memory than SHMTOTAL allows, you have not configured the database server correctly. Lowering the value of BUFFERS or DS_TOTAL_MEMORY is one possible solution, and increasing the value of SHMTOTAL is another.

Processes That Attach to Shared Memory

The following processes attach to the database server shared memory:

- Client-application processes that communicate with the database server through the shared-memory communications portion (ipcshm) ♦
- Database server virtual processors
- Database server utilities

UNIX

The following sections describe how each type of process attaches to the database server shared memory.

How a Client Attaches to the Communications Portion

Client-application processes that communicate with the database server through shared memory (nettype ipcshm) attach transparently to the communications portion of shared memory. System-library functions that are automatically compiled into the application enable it to attach to the communications portion of shared memory. For information on specifying a shared-memory connection, see [Chapter 3, “Client/Server Communications,”](#) and [“Network Virtual Processors”](#) on page 5-31.

If the **INFORMIXSHMBASE** environment variable is not set, the client application attaches to the communications portion at an address that is platform specific. If the client application attaches to other shared-memory segments (not database server shared memory), the user can set the **INFORMIXSHMBASE** environment variable to specify the address at which to attach the database server shared-memory communications segments. When you specify the address at which to address the shared-memory communications segments, you can prevent the database server from colliding with the other shared-memory segments that your application uses. For information on how to set the **INFORMIXSHMBASE** environment variable, refer to the *IBM Informix Guide to SQL: Reference*.

How Utilities Attach to Shared Memory

Database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory through one of the following files.

Operating System	File
UNIX	\$INFORMIXDIR/etc/.infos.servername
Windows	%INFORMIXDIR%\etc\.infos.servername

The variable *servername* is the value of the DBSERVERNAME parameter in the **ONCONFIG** file. The utilities obtain the *servername* portion of the filename from the **INFORMIXSERVER** environment variable.

The **oninit** process reads the ONCONFIG file and creates the file **.infos.servername** when it starts the database server. The file is removed when the database server terminates.

How Virtual Processors Attach to Shared Memory

The database server virtual processors attach to shared memory during initialization. During this process, the database server must satisfy the following two requirements:

- Ensure that all virtual processors can locate and access the same shared-memory segments
- Ensure that the shared-memory segments reside in physical memory locations that are different than the shared-memory segments assigned to other instances of the database server, if any, on the same computer

The database server uses two configuration parameters, **SERVERNUM** and **SHMBASE**, to meet these requirements.

When a virtual processor attaches to shared memory, it performs the following major steps:

- Accesses the **SERVERNUM** parameter from the ONCONFIG file
- Uses **SERVERNUM** to calculate a shared-memory key value
- Requests a shared-memory segment using the shared-memory key value

The operating system returns the shared-memory identifier for the first shared-memory segment.

- Directs the operating system to attach the first shared-memory segment to its process space at **SHMBASE**
- Attaches additional shared-memory segments, if required, to be contiguous with the first segment

The following sections describe how the database server uses the values of the **SERVERNUM** and **SHMBASE** configuration parameters in the process of attaching shared-memory segments.

Obtaining Key Values for Shared-Memory Segments

The values of the SERVERNUM configuration parameter and *shmkey*, an internally calculated number, determine the unique key value for each shared-memory segment.

To see the key values for shared-memory segments, run the **onstat -g seg** command. For more information, see the sections on SHMADD and the buffer pool in your *Performance Guide*.

When a virtual processor requests that the operating system attach the first shared-memory segment, it supplies the unique key value to identify the segment. In return, the operating system passes back a *shared-memory segment identifier* associated with the key value. Using this identifier, the virtual processor requests that the operating system attach the segment of shared memory to the virtual-processor address space.

Specifying Where to Attach the First Shared-Memory Segment

The SHMBASE parameter in the ONCONFIG file specifies the virtual address where each virtual processor attaches the first, or base, shared-memory segment. Each virtual processor attaches to the first shared-memory segment at the same virtual address. This situation enables all virtual processors within the same database server instance to reference the same locations in shared memory without needing to calculate shared-memory addresses. All shared-memory addresses for an instance of the database server are relative to SHMBASE.



Warning: *It is recommended that you not attempt to change the value of SHMBASE.*

The value of SHMBASE is sensitive for the following reasons:

- The specific value of SHMBASE is often computer dependent. It is not an arbitrary number. Informix selects a value for SHMBASE that keeps the shared-memory segments safe when the virtual processor dynamically acquires additional memory space.
- Different operating systems accommodate additional memory at different virtual addresses. Some architectures extend the highest virtual address of the virtual-processor data segment to accommodate the next segment. In this case, the data segment might grow into the shared-memory segment.

UNIX

- Some versions of UNIX require the user to specify a SHMBASE parameter of virtual address zero. The zero address informs the UNIX kernel that the kernel should pick the best address at which to attach the shared-memory segments. However, not all UNIX architectures support this option. Moreover, on some systems, the selection that the kernel makes might not be the best selection. ♦

Attaching Additional Shared-Memory Segments

Each virtual processor must attach to the total amount of shared memory that the database server has acquired. After a virtual processor attaches each shared-memory segment, it calculates how much shared memory it has attached and how much remains. The database server facilitates this process by writing a shared-memory header to the first shared-memory segment. Sixteen bytes into the header, a virtual processor can obtain the following data:

- The total size of shared memory for this database server
- The size of each shared-memory segment

To attach additional shared-memory segments, a virtual processor requests them from the operating system in much the same way that it requested the first segment. For the additional segments, however, the virtual processor adds 1 to the previous value of *shmkey*. The virtual processor directs the operating system to attach the segment at the address that results from the following calculation:

$$\text{SHMBASE} + (\text{seg_size} \times \text{number of attached segments})$$

The virtual processor repeats this process until it has acquired the total amount of shared memory.

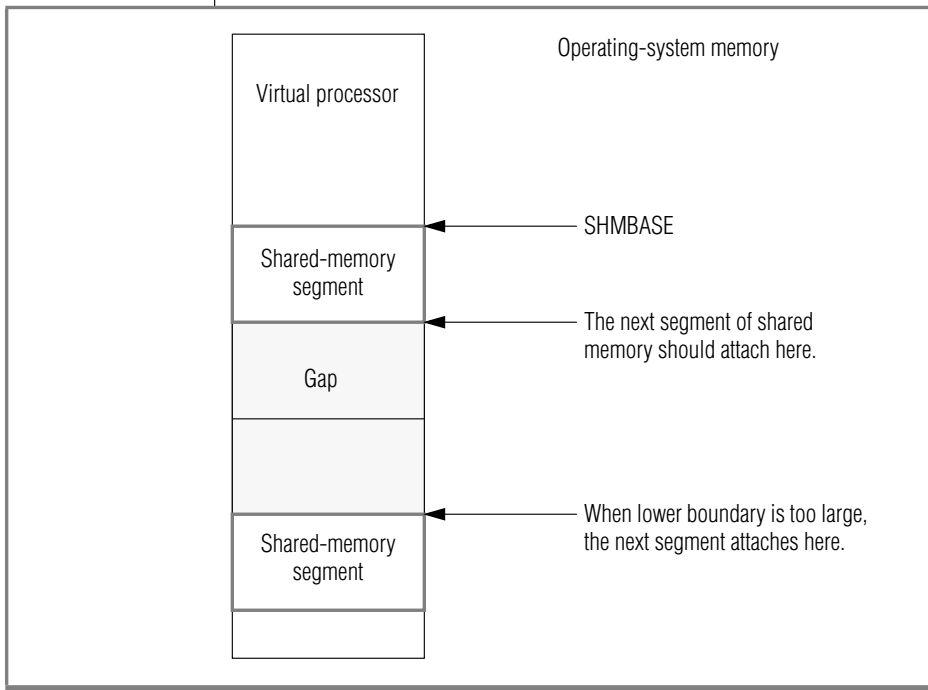
Given the initial key value of $(\text{SERVERNUM} * 65536) + \text{shmkey}$, the database server can request up to 65,536 shared-memory segments before it could request a shared-memory key value used by another database server instance on the same computer.

Defining the Shared-Memory Lower-Boundary Address

If your operating system uses a parameter to define the lower boundary address for shared memory, and the parameter is set incorrectly, it can prevent the shared-memory segments from being attached contiguously.

Figure 7-3 illustrates the problem. If the lower-boundary address is less than the ending address of the previous segment plus the size of the current segment, the operating system attaches the current segment at a point beyond the end of the previous segment. This action creates a gap between the two segments. Because shared memory must be attached to a virtual processor so that it looks like contiguous memory, this gap creates problems. The database server receives errors when this situation occurs. To correct the problem, check the operating-system kernel parameter that specifies the lower-boundary address or reconfigure the kernel to allow larger shared-memory segments. For a description of the operating-system kernel parameter, refer to “[Shared-Memory Lower-Boundary Address](#)” on page 8-5.

Figure 7-3
*Shared-Memory
Lower-Boundary
Address Overview*



Resident Shared-Memory Segments

The operating system, as it switches between the processes running on the system, normally swaps the contents of portions of memory to disk. When a portion of memory is designated as *resident*, however, it is not swapped to disk. Keeping frequently accessed data resident in memory improves performance because it reduces the number of disk I/O operations that would otherwise be required to access that data.

The database server requests that the operating system keep the virtual portions in physical memory when the following two conditions exist:

- The operating system supports shared-memory residency.
- The RESIDENT parameter in the ONCONFIG file is set to -1 or a value that is greater than 0.



Warning: *You must consider the use of shared memory by all applications when you consider whether to set the RESIDENT parameter to -1. Locking all shared memory for the use of the Informix database server can adversely affect the performance of other applications, if any, on the same computer.*

For more information on the RESIDENT configuration parameter, refer to the *Administrator's Reference*.

Resident Portion of Shared Memory

The resident portion of the database server shared memory stores the following data structures that do not change in size while the database server is running:

- Shared-memory header
- Buffer pool
- Logical-log buffer
- Physical-log buffer
- Lock table

Shared-Memory Header

The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool.

The shared-memory header also contains pointers to the locations of these structures. When a virtual processor first attaches to shared memory, it reads address information in the shared-memory header for directions to all other structures.

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

Shared-Memory Buffer Pool

The buffer pool in the resident portion of shared memory contains buffers that store dbspace pages read from disk. The pool of buffers comprises the largest allocation of the resident portion of shared memory.

Figure 7-4 illustrates the shared-memory header and the buffer pool.

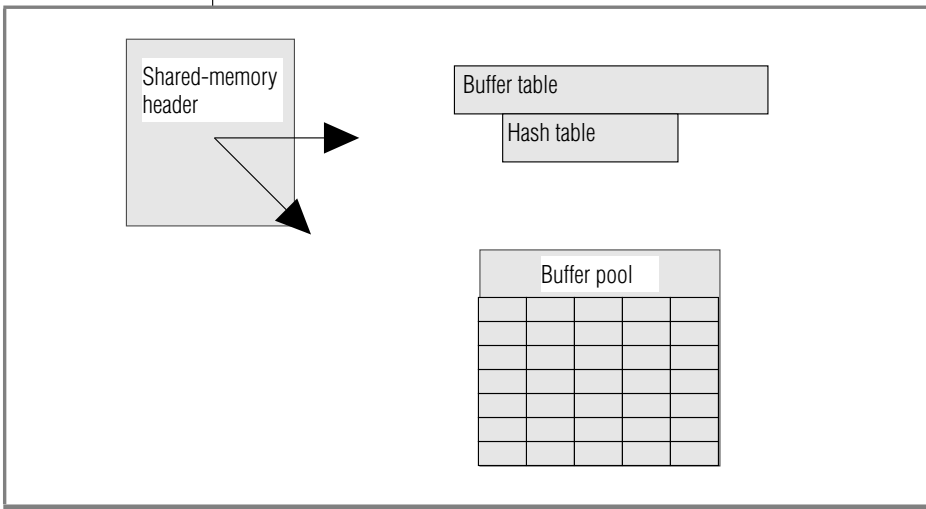


Figure 7-4
*Shared-Memory
Buffer Pool*

You specify the number of buffers in the buffer pool with the `BUFFERS` parameter in the `ONCONFIG` file. `BUFFERS` defaults to 1000 buffers. To allocate the proper number of buffers, start with at least four buffers per user. For more than 500 users, the minimum requirement is 2000 buffers. Too few buffers can severely impact performance, so you must monitor the database server and tune the value of `BUFFERS` to determine an acceptable value. For more information on tuning the number of buffers, refer to your *Performance Guide*.

For more information on setting the `BUFFERS` configuration parameter, refer to the *Administrator's Reference*.

The status of the buffers is tracked through the buffer table. Within shared memory, buffers are organized into LRU `BUFFER QUEUES`. Buffer acquisition is managed through the use of latches, called *mutexes*, and lock-access information.

For a description of how LRU queues work, refer to [“LRU Queues” on page 7-34](#). For a description of mutexes, refer to [“Mutexes” on page 5-19](#).

Buffer Overflow to the Virtual Portion

Because the maximum number of buffers in 64-bit addressing can be as large as $2^{31}-1$, the resident portion of shared memory might not be able to hold all of the buffers in a large buffer pool. In this case, the virtual portion of database server shared memory might hold some of the buffers.

Buffer Size

Each buffer is the size of one database server page. In general, the database server performs I/O in full-page units, the size of a buffer. The exceptions are I/O performed from big buffers, from blob space buffers, or from lightweight I/O buffers. (See [“Big Buffers” on page 7-28](#) and [“Creation of Blobpage Buffers” on page 7-48](#).) For information about when to use private buffers, see the section on lightweight I/O operations in the chapter on configuration effects on I/O utilization in your *Performance Guide*.

The `onstat -b` command displays information about the buffers. For information on `onstat`, refer to the utilities chapter in the *Administrator's Reference*.

Logical-Log Buffer

The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server. The logical log contains the following five types of log records, in addition to many others:

- SQL data definition statements for all databases
- SQL data manipulation statements for databases that were created with logging
- Record of a change to the logging status of a database
- Record of a full or fuzzy checkpoint
- Record of a change to the configuration

The database server uses only one of the logical-log buffers at a time. This buffer is the current logical-log buffer. Before the database server flushes the current logical-log buffer to disk, it makes the second logical-log buffer the current one so that it can continue writing while the first buffer is flushed. If the second logical-log buffer fills before the first one finishes flushing, the third logical-log buffer becomes the current one. This process is illustrated in [Figure 7-5](#).

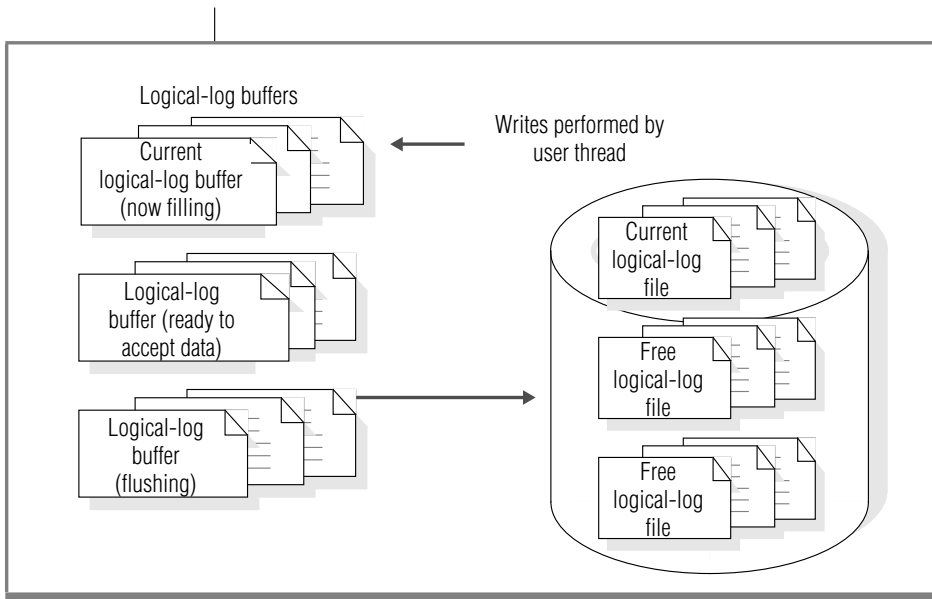


Figure 7-5
The Logical-Log Buffer and Its Relation to the Logical-Log Files on Disk

For a description of how the database server flushes the logical-log buffer, refer to [“Flushing the Logical-Log Buffer” on page 7-45](#).

The LOGBUFF parameter in the ONCONFIG file specifies the size of the logical-log buffers. Small buffers can create problems if you store records larger than the size of the buffers (for example, TEXT or BYTE data in dbspaces). For the possible values that you can assign to this configuration parameter, refer to the *Administrator's Reference*.

For information on the impact of TEXT and BYTE data on shared memory buffers, refer to [“Buffering Large-Object Data” on page 7-46](#).

Physical-Log Buffer

The database server uses the physical-log buffer to hold before-images of some of the modified dbspace pages. The before-images in the physical log and the logical-log records enable the database server to restore consistency to its databases after a system failure.

The physical-log buffer is actually two buffers. Double buffering permits the database server processes to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. For a description of how the database server flushes the physical-log buffer, refer to [“Flushing the Physical-Log Buffer” on page 7-42](#). For information on monitoring the physical-log file, refer to [“Monitoring Physical and Logical-Logging Activity” on page 16-6](#).

The PHYSBUFF parameter in the ONCONFIG file specifies the size of the physical-log buffers. A write to the physical-log buffer writes exactly one page. If the specified size of the physical-log buffer is not evenly divisible by the page size, the database server rounds the size down to the nearest value that is evenly divisible by the page size. Although some operations require the buffer to be flushed sooner, in general the database server flushes the buffer to the physical-log file on disk when the buffer fills. Thus, the size of the buffer determines how frequently the database server needs to flush it to disk. For more information on this configuration parameter, refer to the *Administrator's Reference*.

High-Availability Data-Replication Buffer

High-Availability Data Replication (HDR) requires two instances of the database server, a primary instance and a secondary instance, running on two computers. If you implement HDR for your database server, the database server holds logical-log records in the HDR buffers before it sends them to the secondary database server. An HDR buffer is always the same size as the logical-log buffer. For information on the size of the logical-log buffer, refer to the preceding section, [“Logical-Log Buffer” on page 7-19](#). For more information on how the HDR buffer is used, refer to [“How HDR Works” on page 19-9](#).

Lock Table

A lock is created when a user thread writes an entry in the lock table. The lock table is the pool of available locks. Each lock is 44 bytes. A single transaction can own multiple locks. For an explanation of locking and the SQL statements associated with locking, see the *IBM Informix Guide to SQL: Tutorial*.

The following information, which is stored in the lock table, describes the lock:

- The address of the transaction that owns the lock
- The type of lock (exclusive, update, shared, byte, or intent)
- The page and rowid that is locked
- The tblspace where the lock is placed
- Information about the bytes locked (byte-range locks for smart large objects):
 - Smart-large-object ID
 - Offset into the smart large object where the locked bytes begin
 - The number of bytes locked, starting at the offset

To specify the initial size of the lock table, set the LOCKS configuration parameter.

If the number of locks allocated by sessions exceeds the value of LOCKS, the database server doubles the size of the lock table, up to 15 times. The maximum value of LOCKS is 8,000,000. Use the DEF_TABLE_LOCKMODE configuration parameter to set the lock mode to page or row for new tables.

For more information on using and monitoring locks, see the chapter on locking in your *Performance Guide* and the *IBM Informix Guide to SQL: Tutorial*. For more information on using LOCKS to specify the number of locks for a session, see the chapter on configuration parameters in the *Administrator's Reference* and the chapter on configuration effects on memory utilization in your *Performance Guide*.

Virtual Portion of Shared Memory

The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system. As the database server executes, it automatically attaches additional operating-system segments, as needed, to the virtual portion.

Management of the Virtual Portion of Shared Memory

The database server uses memory *pools* to track memory allocations that are similar in type and size. Keeping related memory allocations in a pool helps to reduce memory fragmentation. It also enables the database server to free a large allocation of memory at one time, as opposed to freeing each piece that makes up the pool.

All sessions have one or more memory pools. When the database server needs memory, it looks first in the specified pool. If insufficient memory is available in a pool to satisfy a request, the database server adds memory from the system pool. If the database server cannot find enough memory in the system pool, it dynamically allocates more segments to the virtual portion.

The database server allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, system catalog, SPL routine caches, SQL statement cache, sort pools, and message buffers) from pools that track free space through a linked list. When the database server allocates a portion of memory, it first searches the pool free-list for a *fragment* of sufficient size. If it finds none, it brings new blocks into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is destroyed. When the database server starts a session for a client application, for example, it allocates memory for the session pool. When the session terminates, the database server returns the allocated memory as free fragments.

Size of the Virtual Portion of Shared Memory

To specify the initial size of the virtual shared-memory portion, set the SHMVIRTSIZE parameter in the ONCONFIG file. To specify the size of segments that are later added to the virtual shared memory, set the SHMADD parameter. To specify the amount of memory available for PDQ queries, set the DS_TOTAL_MEMORY parameter.

For more information on determining the size of virtual shared memory, refer to [“Adding a Segment to the Virtual Portion of Shared Memory” on page 8-14](#) and to the DS_TOTAL_SIZE, SHMVIRTSIZE, and SHMADD configuration parameters in the *Administrator’s Reference*.

Components of the Virtual Portion of Shared Memory

The virtual portion of shared memory stores the following data:

- Internal tables
- Big buffers
- Session data
- Thread data (stacks and heaps)
- Data-distribution cache
- Dictionary cache
- SPL routine cache
- SQL statement cache
- Sorting pool
- Global pool

Shared-Memory Internal Tables

The database server shared memory contains seven internal tables that track shared-memory resources. The shared-memory internal tables are as follows:

- Buffer table
- Chunk table
- Dbspace table
- Page-cleaner table
- Tblspace table
- Transaction table
- User table

Buffer Table

The buffer table tracks the addresses and status of the individual buffers in the shared-memory pool. When a buffer is used, it contains an image of a data or index page from disk. For more information on the purpose and content of a disk page, refer to [“Pages” on page 9-10](#).

Each buffer in the buffer table contains the following control information, which is needed for buffer management:

- Buffer status

Buffer status is described as empty, unmodified, or modified. An unmodified buffer contains data, but the data can be overwritten. A modified (dirty) buffer contains data that must be written to disk before it can be overwritten.

- Current lock-access level

Buffers receive lock-access levels depending on the type of operation that the user thread is executing. The database server supports two buffer lock-access levels: shared and exclusive.

- Threads waiting for the buffer

Each buffer header maintains a list of the threads that are waiting for the buffer and the lock-access level that each waiting thread requires.

Each database server buffer has one entry in the buffer table.

For information on the database server buffers, refer to [“Resident Portion of Shared Memory” on page 7-16](#). For information on how to monitor the buffers, refer to [“Monitoring Buffers” on page 8-17](#).

The database server determines the number of entries in the buffer-table hash table, based on the number of allocated buffers. The maximum number of hash values is the largest power of 2 that is less than the value of BUFFERS.

Chunk Table

The chunk table tracks all chunks in the database server. If mirroring has been enabled, a corresponding mirrored chunk table is also created when shared memory is initialized. The mirrored chunk table tracks all mirrored chunks.

The chunk table in shared memory contains information that enables the database server to locate chunks on disk. This information includes the number of the initial chunk and the number of the next chunk in the dbspace. Flags also describe chunk status: mirror or primary; offline, online, or recovery mode; and whether this chunk is part of a blob space. For information on monitoring chunks, refer to [“Monitoring Chunks” on page 10-41](#).

The maximum number of entries in the chunk table might be limited by the maximum number of file descriptors that your operating system allows per process. You can usually specify the number of file descriptors per process with an operating-system kernel-configuration parameter. For details, consult your operating-system manuals.

Dbospace Table

The dbospace table tracks storage spaces in the database server. The dbospace-table information includes the following information about each dbospace:

- Dbospace number
- Dbospace name and owner
- Dbospace mirror status (mirrored or not)
- Date and time that the dbospace was created

If the storage space is a blobospace, flags indicate the media where the blobospace is located: magnetic, removable, or optical media. If the storage space is an sbospace, it contains internal tables that track metadata for smart large objects and large contiguous blocks of pages containing user data.

For information on monitoring dbospaces, refer to [“Monitoring Disk Usage” on page 10-41](#).

Page-Cleaner Table

The page-cleaner table tracks the state and location of each of the page-cleaner threads. The number of page-cleaner threads is specified by the CLEANERS configuration parameter in the ONCONFIG file. For advice on how many page-cleaner threads to specify, refer to the chapter on configuration parameters in the *Administrator's Reference*.

The page-cleaner table always contains 128 entries, regardless of the number of page-cleaner threads specified by the CLEANERS parameter in the ONCONFIG file.

For information on monitoring the activity of page-cleaner threads, refer to the **onstat -F** option in the utilities chapter of the *Administrator's Reference*.

Tblspace Table

The tblspace table tracks all active tblspaces in a database server instance. An active tblspace is one that is currently in use by a database session. Each active table accounts for one entry in the tblspace table. Active tblspaces include database tables, temporary tables, and internal control tables, such as system catalog tables. Each tblspace table entry includes header information about the tblspace, the tblspace name, and pointers to the tblspace tblspace in dbspaces on disk. (The shared-memory active tblspace table is different from the tblspace tblspace.) For information on monitoring tblspaces, refer to [“Monitoring Tblspaces and Extents” on page 10-48](#).

The database server manages one tblspace table for each dbspace.

Transaction Table

The transaction table tracks all transactions in the database server.

Tracking information derived from the transaction table appears in the **onstat -x** display. For an example of the output that **onstat -x** displays, refer to monitoring transactions in your *Performance Guide*.

The database server automatically increases the number of entries in the transaction table, up to a maximum of 32,767, based on the number of current transactions.

For more information on transactions and the SQL statements that you use with transactions, refer to the *IBM Informix Guide to SQL: Tutorial*, the *IBM Informix Guide to SQL: Reference*, and the *IBM Informix Guide to SQL: Syntax*.

The transaction table also specifically supports the X/Open environment. Support for the X/Open environment requires TP/XA. For a description of a transaction in this environment, refer to the *TP/XA Programmer's Manual*. ♦

User Table

The user table tracks all user threads and system threads. Each client session has one primary thread and zero-to-many secondary threads, depending on the level of parallelism specified. System threads include one to monitor and control checkpoints, one to process **onmode** commands, the B-tree scanner threads, and page-cleaner threads.

The database server increases the number of entries in the user table as needed. You can monitor user threads with the **onstat -u** command.

Big Buffers

A big buffer is a single buffer that is made up of several pages. The actual number of pages is platform dependent. The database server allocates big buffers to improve performance on large reads and writes.

The database server uses a big buffer whenever it writes to disk multiple pages that are physically contiguous. For example, the database server tries to use a big buffer to perform a series of sequential reads (light scans) or to read into shared memory simple large objects that are stored in a dbspace.

Users do not have control over the big buffers. If the database server uses light scans, it allocates big buffers from shared memory.

For information on monitoring big buffers with the **onstat** command, refer to the chapter on configuration effects on I/O activity in your *Performance Guide*.

Session Data

When a client application requests a connection to the database server, the database server begins a *session* with the client and creates a data structure for the session in shared memory called the *session-control block*. The session-control block stores the session ID, the user ID, the process ID of the client, the name of the host computer, and various status flags.

The database server allocates memory for session data as needed.

Thread Data

When a client connects to the database server, in addition to starting a session, the database server starts a primary session thread and creates a *thread-control block* for it in shared memory.

The database server also starts internal threads on its own behalf and creates thread-control blocks for them. When the database server switches from running one thread to running another one (a context switch), it saves information about the thread—such as the register contents, program counter (address of the next instruction), and global pointers—in the thread-control block. For more information on the thread-control block and how it is used, refer to [“Context Switching” on page 5-13](#).

The database server allocates memory for thread-control blocks as needed.

Stacks

Each thread in the database server has its own stack area in the virtual portion of shared memory. For a description of how threads use stacks, refer to [“Stacks” on page 5-15](#). For information on how to monitor the size of the stack for a session, refer to monitoring sessions and threads section in your *Performance Guide*.

The size of the stack space for user threads is specified by the STACKSIZE parameter in the ONCONFIG file. The default size of the stack is 32 kilobytes. You can change the size of the stack for all user threads, if necessary, by changing the value of STACKSIZE. For information and a warning on setting the size of the stack, refer to STACKSIZE in the chapter on configuration parameters in the *Administrator's Reference*.

To alter the size of the stack for the primary thread of a specific session, set the **INFORMIXSTACKSIZE** environment variable. The value of **INFORMIXSTACKSIZE** overrides the value of STACKSIZE for a particular user. For information on how to override the stack size for a particular user, refer to the description of the **INFORMIXSTACKSIZE** environment variable in the *IBM Informix Guide to SQL: Reference*.

To more safely alter the size of stack space, use the **INFORMIXSTACKSIZE** environment variable rather than alter the configuration parameter STACKSIZE. The **INFORMIXSTACKSIZE** environment variable affects the stack space for only one user, and it is less likely to affect new client applications that initially were not measured.

Heaps

Each thread has a heap to hold data structures that it creates while it is running. A heap is dynamically allocated when the thread is created. The size of the thread heap is not configurable.

Data-Distribution Cache

The database server uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost. When the database server accesses the distribution statistics for a specific column the first time, it reads the distribution statistics from the **sysdistrib** system catalog table on disk and stores the statistics in the data-distribution cache. These statistics can then be read for the optimization of subsequent queries that access the column.

Performance improves if these statistics are efficiently stored and accessed from the data-distribution cache. You can configure the size of the data-distribution cache with the DS_HASHSIZE and DS_POOLSIZE configuration parameters. For information about changing the default size of the data-distribution cache, refer to the chapter on queries and the query optimizer in your *Performance Guide*.

Dictionary Cache

When a session executes an SQL statement that requires access to a system catalog table, the database server reads the system catalog tables and stores them in structures that it can access more efficiently. These structures are created in the virtual portion of shared memory for use by all sessions. These structures constitute the dictionary cache.

You can configure the size of the dictionary cache with the DD_HASHSIZE and DD_HASHMAX configuration parameters. For more information about these parameters, refer to the chapter on configuration effects on memory in your *Performance Guide*.

SQL Statement Cache

The SQL statement cache reduces memory usage and preparation time for queries. The database server uses the SQL statement cache to store parsed and optimized SQL statements that a user executes. When users execute a statement stored in the SQL statement cache, the database server does not parse and optimize the statement again, so performance improves.

For more information, see [“Setting SQL Statement Cache Parameters” on page 8-11](#). For details on how these parameters affect the performance of the SQL statement cache, refer to the *Performance Guide*.

Sorting Memory

The following database operations can use large amounts of the virtual portion of shared memory to sort data:

- Decision-support queries that involve joins, groups, aggregates and sort operations
- Index builds
- UPDATE STATISTICS statement in SQL

The amount of virtual shared memory that the database server allocates for a sort depends on the number of rows to be sorted and the size of the row, along with other factors.

For information on parallel sorts, refer to your *Performance Guide*.

SPL Routine and the UDR Cache

The database server converts an SPL routine to executable format and stores the routine in the UDR cache, where it can be accessed by any session.

When a session needs to access an SPL routine or other user-defined routine for the first time, the database server reads the definition from the system catalog tables and stores the definition in the UDR cache.

You can configure the size of the UDR cache with the PC_HASHSIZE and PC_POOLSIZE configuration parameters. For information about changing the default size of the UDR cache, refer to the chapter on queries and the query optimizer in your *Performance Guide*.

Global Pool

The global pool stores structures that are global to the database server. For example, the global pool contains the message queues where poll threads for network communications deposit messages from clients. The **sqlexec** threads pick up the messages from the global pool and process them.

For more information, see the sections on network buffer pools and virtual portion of shared memory in your *Performance Guide*.

Communications Portion of Shared Memory

The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you initialize shared memory. The communications portion contains the message buffers for local client applications that use shared memory to communicate with the database server.

The size of the communications portion of shared memory equals approximately 12 kilobytes multiplied by the expected number of connections needed for shared-memory communications (**nettype ipcshm**). If **nettype ipcshm** is not present, the expected number of connections defaults to 50. For information about how a client attaches to the communications portion of shared memory, refer to [“How a Client Attaches to the Communications Portion” on page 7-11](#).

Virtual-Extension Portion of Shared Memory

The virtual-extension portion of shared memory contains additional virtual segments and virtual-extension segments. Virtual-extension segments contain thread heaps for DataBlade modules and user-defined routines that run in user-defined virtual processors.

The SHMADD and SHMTOTAL configuration parameters apply to the virtual-extension portion of shared memory, just as they do to the other portions of shared memory.

Concurrency Control

The database server threads that run on the same virtual processor and on separate virtual processors share access to resources in shared memory. When a thread writes to shared memory, it uses mechanisms called *mutexes* and *locks* to prevent other threads from simultaneously writing to the same area. A mutex gives a thread the right to access a shared-memory resource. A lock prevents other threads from writing to a buffer until the thread that placed the lock is finished with the buffer and releases the lock.

Shared-Memory Mutexes

The database server uses *mutexes* to coordinate threads as they attempt to modify data in shared memory. Every modifiable shared-memory resource is associated with a mutex. Before a thread can modify a shared-memory resource, it must first acquire the mutex associated with that resource. After the thread acquires the mutex, it can modify the resource. When the modification is complete, the thread releases the mutex.

If a thread tries to obtain a mutex and finds that it is held by another thread, the incoming thread must wait for the mutex to be released.

For example, two threads can attempt to access the same slot in the chunk table, but only one can acquire the mutex associated with the chunk table. Only the thread that holds the mutex can write its entry in the chunk table. The second thread must wait for the mutex to be released and then acquire it.

For information on monitoring mutexes (which are also referred to as latches), refer to [“Monitoring the Shared-Memory Profile and Latches” on page 8-15](#).

Shared-Memory Buffer Locks

A primary benefit of shared memory is the ability of database server threads to share access to disk pages stored in the shared-memory buffer pool. The database server maintains thread isolation while it achieves this increased concurrency through a strategy for locking the data buffers.

Types of Buffer Locks

The database server uses two types of locks to manage access to shared-memory buffers:

- Share locks
- Exclusive locks

Each of these lock types enforces the required level of thread isolation during execution.

Share Lock

A buffer is in share mode, or has a share lock, if multiple threads have access to the buffer to read the data but none intends to modify the data.

Exclusive Lock

A buffer is in exclusive mode, or has an exclusive lock, if a thread demands exclusive access to the buffer. All other thread requests that access the buffer are placed in the wait queue. When the executing thread is ready to release the exclusive lock, it wakes the next thread in the wait queue.

Database Server Thread Access to Shared Buffers

Database server threads access shared buffers through a system of queues, using mutexes and locks to synchronize access and protect data.

LRU Queues

A buffer holds data for the purpose of caching. The database server uses the least-recently used (LRU) queues to replace the cached data.

The LRUS parameter in the ONCONFIG file specifies the number of LRU queues to create when database server shared memory is initialized. You can tune the value of the LRUS, LRU_MIN_DIRTY, and LRU_MAX_DIRTY parameters, to control how frequently the shared-memory buffers are flushed to disk.

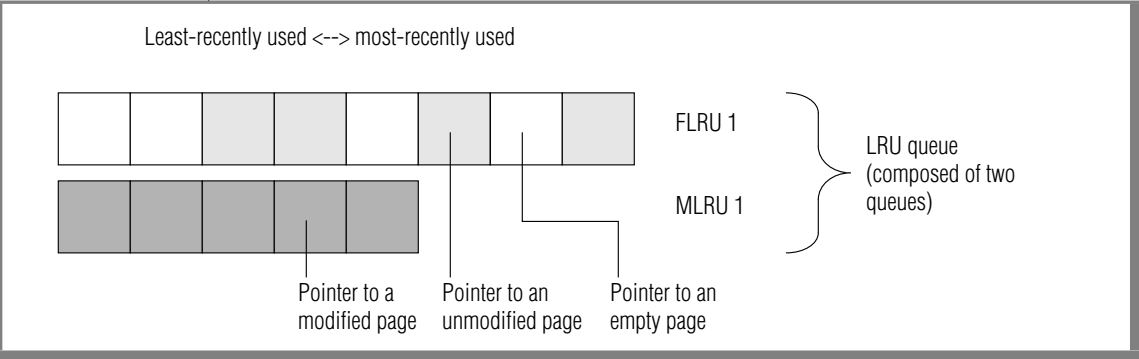
Components of LRU Queue

Each LRU queue is composed of a pair of linked lists, as follows:

- FLRU (free least-recently used) list, which tracks free or unmodified pages in the queue
- MLRU (modified least-recently used) list, which tracks modified pages in the queue

The free or unmodified page list is referred to as the FLRU queue of the queue pair, and the modified page list is referred to as the MLRU queue. The two separate lists eliminate the need to search a queue for a free or unmodified page. Figure 7-6 illustrates the structure of the LRU queues.

Figure 7-6
LRU Queue



Pages in Least-Recently Used Order

When the database server processes a request to read a page from disk, it must decide which page to replace in memory. Rather than select a page randomly, the database server assumes that recently referenced pages are more likely to be referenced in the future than pages that it has not referenced for some time. Thus, rather than replacing a recently accessed page, the database server replaces a least-recently accessed page. By maintaining pages in least-recently to most-recently used order, the database server can easily locate the least-recently used pages in memory.

LRU Queues and Buffer-Pool Management

Before processing begins, all page buffers are empty, and every buffer is represented by an entry in one of the FLRU queues. The buffers are evenly distributed among the FLRU queues. To calculate the number of buffers in each queue, divide the total number of buffers (BUFFERS parameter) by the number of LRU queues (LRUS parameter).

When a user thread needs to acquire a buffer, the database server randomly selects one of the FLRU queues and uses the oldest or least-recently used entry in the list. If the least-recently used page can be latched, that page is removed from the queue.

If the FLRU queue is locked, and the end page cannot be latched, the database server randomly selects another FLRU queue.

If a user thread is searching for a specific page in shared memory, it obtains the LRU-queue location of the page from the control information stored in the buffer table.

After an executing thread finishes its work, it releases the buffer. If the page has been modified, the buffer is placed at the most-recently used end of an MLRU QUEUE. If the page was read but not modified, the buffer is returned to the FLRU queue at its most-recently used end. For information on how to monitor LRU queues, refer to [“Monitoring Buffer-Pool Activity” on page 8-20](#).

Number of LRU Queues to Configure

Multiple LRU queues have two purposes:

- They reduce user-thread contention for the queues.
- They allow multiple cleaners to flush pages from LRU queues and maintain the percentage of dirty pages at an acceptable level.

Initial values for the LRUS parameter are recommended based on the number of CPUs that are available on your computer. If your computer is a uniprocessor, start by setting the LRUS parameter to 4. If your computer is a multiprocessor, use the following formula:

$$\text{LRUS} = \max(4, (\text{NUMCPUVPS}))$$

After you provide an initial value to the LRUS parameter, monitor your LRU queues with `onstat -R`. If you find that the percent of dirty LRU queues consistently exceeds the value of the `LRU_MAX_DIRTY` parameter, increase the value of the LRUS configuration parameter to add more LRU queues.

For example, suppose you set `LRU_MAX_DIRTY` to 70 and find that your LRU queues are consistently 75 percent dirty. Consider increasing the value of the LRUS configuration parameter. If you increase the number of LRU queues, you shorten the length of the queues, thereby reducing the work of the page cleaners. However, you must allocate a sufficient number of page cleaners with the `CLEANERS` configuration parameter, as discussed in the following section.

Number of Cleaners to Allocate

In general, it is recommended that you configure one cleaner for each disk that your applications update frequently. However, you should also consider the length of your LRU queues and frequency of checkpoints, as explained in the following paragraphs.

In addition to insufficient LRU queues, another factor that influences whether page cleaners keep up with the number of pages that require cleaning is whether you have enough page-cleaner threads allocated. The percent of dirty pages might exceed `LRU_MAX_DIRTY` in some queues because no page cleaners are available to clean the queues. After a while, the page cleaners might be too far behind to catch up, and the buffer pool becomes dirtier than the percent that you specified in `LRU_MAX_DIRTY`.

For example, suppose that the `CLEANERS` parameter is set to 8, and you increase the number of LRU queues from 8 to 12. You can expect little in the way of a performance gain because the 8 cleaners must now share the work of cleaning an additional 4 queues. If you increase the number of `CLEANERS` to 12, each of the now-shortened queues can be more efficiently cleaned by a single cleaner.

Setting `CLEANERS` too low can cause performance to suffer whenever a checkpoint occurs because page cleaners must flush all modified pages to disk during checkpoints. If you do not configure a sufficient number of page cleaners, checkpoints take longer, causing overall performance to suffer.

For more information, see [“Flushing Buffer-Pool Buffers”](#) on page 7-41.

Number of Pages Added to the MLRU Queues

Periodically, the page-cleaner threads flush the modified buffers in an MLRU queue to disk. To specify the point at which cleaning begins, use the LRU_MAX_DIRTY configuration parameter.

By specifying when page cleaning begins, the LRU_MAX_DIRTY configuration parameter limits the number of page buffers that can be appended to an MLRU queue. The initial setting of LRU_MAX_DIRTY is 60.00, so page cleaning begins when 60 percent of the buffers managed by a queue are modified.

In practice, page cleaning begins under several conditions, only one of which is when an MLRU queue reaches the value of LRU_MAX_DIRTY. For more information on how the database server performs buffer-pool flushing, refer to [“Flushing Data to Disk” on page 7-40](#).

[Figure 7-7](#) shows how the value of LRU_MAX_DIRTY is applied to an LRU queue to specify when page cleaning begins and thereby limit the number of buffers in an MLRU queue.

```
BUFFERS specified as 8000
LRUS specified as 8
LRU_MAX_DIRTY specified as 60 percent

Page cleaning begins when the number of buffers in the MLRU
queue is equal to LRU_MAX_DIRTY.

Buffers per LRU queue = (8000/8) = 1000

Max buffers in MLRU queue and point at which page cleaning
begins: 1000 x 0.60 = 600
```

Figure 7-7
*How
LRU_MAX_DIRTY
Initiates Page
Cleaning to Limit
the Size of the
MLRU Queue*

End of MLRU Cleaning

You can also specify the point at which MLRU cleaning can end. The LRU_MIN_DIRTY configuration parameter specifies the acceptable percent of buffers in an MLRU queue. The initial setting of LRU_MIN_DIRTY is 50.00, meaning that page cleaning is no longer required when 50 percent of the buffers in an LRU queue are modified. In practice, page cleaning can continue beyond this point as directed by the page-cleaner threads.

Figure 7-8 shows how the value of LRU_MIN_DIRTY is applied to the LRU queue to specify the acceptable percent of buffers in an MLRU queue and the point at which page cleaning ends.

```
BUFFERS specified as 8000
LRUS specified as 8
LRU_MIN_DIRTY specified as 50 percent
```

```
The acceptable number of buffers in the MLRU queue and
the point at which page cleaning can end is equal
to LRU_MIN_DIRTY.
```

```
Buffers per LRU queue = (8000/8) = 1000
```

```
Acceptable number of buffers in MLRU queue and the point
at which page cleaning can end: 1000 x .50 = 500
```

Figure 7-8
*How
LRU_MIN_DIRTY
Specifies the Point
at Which Page
Cleaning Can End*

The LRU_MAX_DIRTY and the LRU_MIN_DIRTY parameters accept decimal numbers. For example, if you set the LRU_MAX_DIRTY to 1.0333 and LRU_MIN_DIRTY to 1.0 this triggers LRU to write at 3,100 dirty buffers and to stop at 3,000 dirty buffers.

For more information on how the database server flushes the buffer pool, refer to [“Flushing Data to Disk” on page 7-40](#).

Configuring the Database Server to Read Ahead

For sequential table or index scans, you can configure the database server to read several pages ahead while the current pages are being processed. A read-ahead enables applications to run faster because they spend less time waiting for disk I/O.

The database server performs a read-ahead whenever it detects the need for it during sequential data or index reads.

The RA_PAGES parameter in the ONCONFIG file specifies the number of pages to read from disk or the index when the database server performs a read-ahead.

The RA_THRESHOLD parameter specifies the number of unprocessed pages in memory that cause the database server to perform another read-ahead. For example, if RA_PAGES is 10, and RA_THRESHOLD is 4, the database server reads ahead 10 pages when 4 pages remain to be processed in the buffer.

For an example of the output that the **onstat -p** command produces to enable you to monitor the database server use of read-ahead, refer to [“Monitoring the Shared-Memory Profile and Latches” on page 8-15](#) and to the utilities chapter in the *Administrator's Reference*.

Database Server Thread Access to Buffer Pages

The database server uses shared-lock buffering to allow more than one database server thread to access the same buffer concurrently in shared memory. The database server uses two categories of buffer locks to provide this concurrency without a loss in thread isolation. The two categories of lock access are *share* and *exclusive*. (For more information, refer to [“Types of Buffer Locks” on page 7-34.](#))

Flushing Data to Disk

Writing a buffer to disk is called *buffer flushing*. When a user thread modifies data in a buffer, it marks the buffer as *dirty*. When the database server flushes the buffer to disk, it subsequently marks the buffer as *not dirty* and allows the data in the buffer to be overwritten.

The database server flushes the following buffers:

- Buffer pool (covered in this section)
- Physical-log buffer
See [“Flushing the Physical-Log Buffer” on page 7-42.](#)
- Logical-log buffer
See [“Flushing the Logical-Log Buffer” on page 7-45.](#)

Page-cleaner threads manage buffer flushing. The database server always runs at least one page-cleaner thread. If the database server is configured for more than one page-cleaner thread, the LRU queues are divided among the page cleaners for more efficient flushing. For information on specifying how many page-cleaner threads the database server runs, refer to the CLEANERS configuration parameter in the *Administrator's Reference*.

Flushing the physical-log buffer, the modified shared-memory page buffers, and the logical-log buffer must be synchronized with page-cleaner activity according to specific rules designed to maintain data consistency.

Flushing Buffer-Pool Buffers

Flushing of the buffers is initiated by any one of the following three conditions:

- The number of buffers in an MLRU queue reaches the number specified by LRU_MAX_DIRTY.
- The page-cleaner threads cannot keep up. In other words, a user thread needs to acquire a buffer, but no unmodified buffers are available.
- The database server needs to execute a checkpoint. (See [“Checkpoints” on page 15-11.](#))

Flushing Before-Images First

The overriding rule of buffer flushing is that the before-images of modified pages are flushed to disk before the modified pages themselves.

In practice, the physical-log buffer is flushed first and then the buffers that contain modified pages. Therefore, even when a shared-memory buffer page needs to be flushed because a user thread is trying to acquire a buffer but none is available (a foreground write), the buffer pages cannot be flushed until the before-image of the page has been written to disk.

Flushing the Physical-Log Buffer

The database server temporarily stores before-images of some of the modified disk pages in the physical-log buffer. If the before-image has been written to the physical-log buffer but not to the physical log on disk, the physical-log buffer must be flushed to disk before the modified page can be flushed to disk. This action is required for the fast-recovery feature.

Both the physical-log buffer and the physical log contribute toward maintaining the physical and logical consistency of the data. For a description of physical logging, refer to [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”](#) For a description of checkpoints and fast recovery, refer to [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”](#)

The following events cause the current physical-log buffer to flush:

- The current physical-log buffer becomes full.
- A modified page in shared memory must be flushed, but the before-image is still in the current physical-log buffer.
- A full or fuzzy checkpoint occurs.

The contents of the physical-log buffer must always be flushed to disk before any data buffers. This rule is required for the fast-recovery feature.

The database server uses only one of the two physical-log buffers at a time. This buffer is the current physical-log buffer. Before the database server flushes the current physical-log buffer to disk, it makes the other buffer the current buffer so that it can continue writing while the first buffer is being flushed.

Synchronizing Buffer Flushing

When shared memory is first initialized, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.

Describing Flushing Activity

To provide you with information about the specific condition that prompted buffer-flushing activity, the database server defines three types of writes and counts how often each write occurs:

- Foreground write
- LRU write
- Chunk write

To display the write counts that the database server maintains, use `onstat -F` as described in the utilities chapter of the *Administrator's Reference*.

If you implement mirroring for the database server, data is always written to the primary chunk first. The write is then repeated on the mirrored chunk. Writes to a mirrored chunk are included in the counts. For more information on monitoring the types of writes that the database server performs, refer to [“Monitoring Buffer-Pool Activity” on page 8-20](#).

Foreground Write

Whenever an **sqlexec** thread writes a buffer to disk, it is termed a *foreground* write. A foreground write occurs when an **sqlexec** thread searches through the LRU queues on behalf of a user but cannot locate an empty or unmodified buffer. To make space, the **sqlexec** thread flushes pages, one at a time, to hold the data to be read from disk. (For more information, refer to [“LRU Queues” on page 7-34](#).)

If the **sqlexec** thread must perform buffer flushing just to acquire a shared-memory buffer, performance can suffer. Foreground writes should be avoided. To display a count of the number of foreground writes, run **onstat -F**. If you find that foreground writes are occurring on a regular basis, tune the value of the page-cleaning parameters. Either increase the number of page cleaners or decrease the value of `LRU_MAX_DIRTY`.

LRU Write

Unlike foreground writes, LRU writes are performed by page cleaners rather than by **sqlexec** threads. The database server performs LRU writes as background writes that typically occur when the percentage of dirty buffers exceeds the percent you that specified in the LRU_MAX_DIRTY configuration parameter.

In addition, a foreground write can trigger an LRU write. When a foreground write occurs, the **sqlexec** thread that performed the write alerts a page-cleaner to wake up and clean the LRU for which it performed the foreground write.

In a properly tuned system, page cleaners ensure that enough unmodified buffer pages are available for storing pages to be read from disk. Thus, **sqlexec** threads that perform a query do not need to flush a page to disk before they read in the disk pages required by the query. This condition can result in significant performance gains for queries that do not make use of foreground writes.

LRU writes are preferred over foreground writes because page-cleaner threads perform buffer writes much more efficiently than **sqlexec** threads do. To monitor both types of writes, use **onstat -F**.

Chunk Write

Chunk writes are commonly performed by page-cleaner threads during a checkpoint or, possibly, when every page in the shared-memory buffer pool is modified. Chunk writes, which are performed as sorted writes, are the most efficient writes available to the database server.

During a chunk write, each page-cleaner thread is assigned to one or more chunks. Each page-cleaner thread reads through the buffer headers and creates an array of pointers to pages that are associated with its specific chunk. (The page cleaners have access to this information because the chunk number is contained within the physical page number address, which is part of the page header.) This sorting minimizes head movement (disk seek time) on the disk and enables the page-cleaner threads to use the big buffers during the write, if possible.

In addition, because user threads must wait for the checkpoint to complete, the page-cleaner threads are not competing with a large number of threads for CPU time. As a result, the page-cleaner threads can finish their work with less context switching.

Flushing the Logical-Log Buffer

The database server uses the shared-memory logical-log buffer as temporary storage for records that describe modifications to database server pages. From the logical-log buffer, these records of changes are written to the current logical-log file on disk and eventually to the logical-log backup media. For a description of logical logging, refer to [Chapter 13, “Logical Log.”](#)

Five events cause the current logical-log buffer to flush:

- The current logical-log buffer becomes full.
- A transaction is prepared or committed in a database with unbuffered logging.
- A nonlogging database session terminates.
- A checkpoint occurs.
- A page is modified that does not require a before-image in the physical log.

The following sections discuss each of these events in detail.

After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging

The following log records cause flushing of the logical-log buffers in a database with unbuffered logging:

- COMMIT
- PREPARE
- XPREPARE
- ENDTRANS

For a comparison of buffered versus unbuffered logging, refer to the SET LOG statement in the *IBM Informix Guide to SQL: Syntax*.

When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates

Even for nonlogging databases, the database server logs certain activities that alter the database schema, such as the creation of tables or extents. When the database server terminates sessions that use unbuffered logging or nonlogging databases, the logical-log buffer is flushed to make sure that any logging activity is recorded.

When a Checkpoint Occurs

For a detailed description of the events that occur during a checkpoint, refer to [“Checkpoints” on page 15-11](#).

When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File

When a page is modified that does not require a before-image in the physical log, the logical-log buffer must be flushed before that page is flushed to disk.

Buffering Large-Object Data

Simple large objects (TEXT or BYTE data) can be stored in either dbspaces or blobspaces. Smart large objects (CLOB or BLOB data) are stored only in sbspaces. The database server uses different methods to access each type of storage space. The following sections describe buffering methods for each.

Writing Simple Large Objects

The database server writes simple large objects to disk pages in a dbspace in the same way that it writes any other data type. For more information, refer to [“Flushing Data to Disk” on page 7-40](#).

You can also assign simple large objects to a blobspace. The database server writes simple large objects to a blobspace differently from the way that it writes other data to a shared-memory buffer and then flushes it to disk. For a description of blobspaces, refer to the chapter on disk structure and storage in the *Administrator's Reference*.

Blobpages and Shared Memory

Blobspace blobpages store large amounts of data. Consequently, the database server does not create or access blobpages by way of the shared-memory buffer pool, and it does not write blobspace blobpages to either the logical or physical logs.

If blobspace data passed through the shared-memory pool, it might dilute the effectiveness of the pool by driving out index pages and data pages. Instead, blobpage data is written directly to disk when it is created.

To reduce logical-log and physical-log traffic, the database server writes blobpages from magnetic media to dbspace backup tapes and logical-log backup tapes in a different way than it writes dbspace pages. For a description of how blobspaces are logged, refer to [“Logging Blobspaces and Simple Large Objects” on page 13-11](#).

Blobpages stored on optical media are not written to dbspace and logical-log backup tapes due to the high reliability of optical media.

Creation of Simple Large Objects

When simple-large-object data is written to disk, the row to which it belongs might not exist yet. During an insert, for example, the simple large object is transferred before the rest of the row data. After the simple large object is stored, the data row is created with a 56-byte descriptor that points to its location. For a description of how simple large objects are stored physically, refer to the structure of a dbspace blobpage in the disk storage and structure chapter of the *Administrator's Reference*.

Creation of Blobpage Buffers

To receive simple large object data from the application process, the database server creates a pair of blobspace buffers, one for reading and one for writing, each the size of one blobpage. Each user has only one set of blobpage buffers and, therefore, can access only one simple large object at a time.

Simple large object data is transferred from the client-application process to the database server in 1-kilobyte segments. The database server begins filling the blobpage buffers with the 1-kilobyte pieces and attempts to buffer two blobpages at a time. The database server buffers two blobpages so that it can determine when to add a forwarding pointer from one page to the next. When it fills the first buffer and discovers that more data remains to transfer, it adds a forward pointer to the next page before it writes the page to disk. When no more data remains to transfer, the database server writes the last page to disk without a forward pointer.

When the thread begins writing the first blobpage buffer to disk, it attempts to perform the I/O based on the user-defined blobpage size. For example, if the blobpage size is 32 kilobytes, the database server attempts to read or write the data in 32,768-byte increments. If the underlying hardware (such as the disk controller) cannot transfer this amount of data in a single operation, the operating-system kernel loops internally (in kernel mode) until the transfer is complete.

The blobpage buffers remain until the thread that created them is finished. When the simple large object is written to disk, the database server deallocates the pair of blobpage buffers. [Figure 7-9](#) illustrates the process of writing a simple large object to a blobpage.

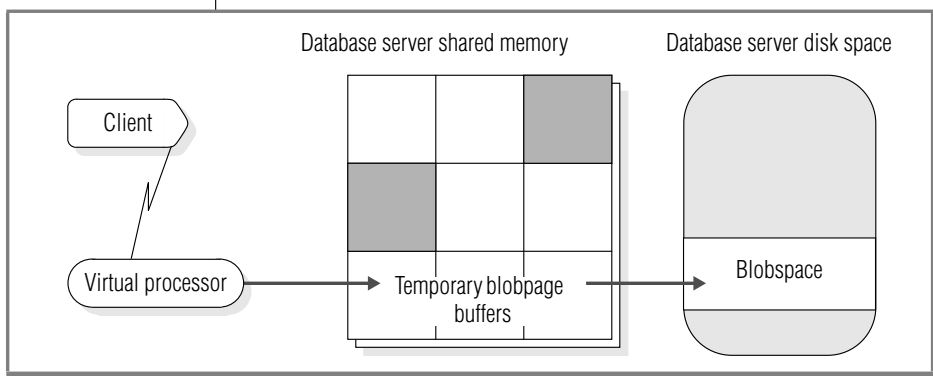


Figure 7-9
Writing Simple
Large Object to a
Blobpage

Blobspace blobpages are allocated and tracked with the free-map page. Links that connect the blobpages and pointers to the next blobpage segments are created as needed.

A record of the operation (insert, update, or delete) is written to the logical-log buffer.

Accessing Smart Large Objects

The database server accesses smart large objects through the shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. However, the user-data portion of a smart large object is buffered at a lower priority than normal buffer pages to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

A smart large object is stored in an sbpace. You cannot store simple large objects in an sbpace, and you cannot store smart large objects in a blobspace. An sbpace consists of a user-data area and a metadata area. The user-data area contains the smart-large-object data. The metadata area contains information about the content of the sbpace. For more information on sbspaces, refer to [“Sbspaces” on page 9-21](#).

Because smart large objects pass through the shared-memory buffer pool and can be logged, you must consider them when you allocate buffers. Use the `BUFFERS` configuration parameter to allocate shared-memory buffers. As a general rule, try to have enough buffers to hold two smart-large-object pages for each concurrently open smart large object. (The additional page is available for read-ahead purposes.) For more information about tuning buffers for smart large objects, refer to your *Performance Guide*.

Use the `LOGBUFF` configuration parameter to specify the size of the logical-log buffer. For information about setting each of the following configuration parameters, refer to the *Administrator's Reference*:

- `BUFFERS`
- `LOGBUFF`

The user-data area of smart large objects that are logged does not pass through the physical log, so the `PHYSBUFF` parameter need not be adjusted for smart large objects.

For more information on the structure of an sbspace, refer to sbspace structure in the disk structures and storage chapter of the *Administrator's Reference*. For information on creating an sbspace, refer to **onspaces** in the utilities chapter of the *Administrator's Reference*.

Memory Use on 64-Bit Platforms

With 64-bit addressing, you can have larger buffer pools to reduce the amount of I/O operations to obtain data from disks. Because 64-bit platforms allow for larger memory-address space, the maximum values for the following memory-related configuration parameters are larger on 64-bit platforms:

- BUFFERS
- CLEANERS
- DS_MAX_QUERIES
- DS_TOTAL_MEMORY
- LOCKS
- LRUS
- SHMADD
- SHMVIRTSIZE

The machine notes for each 64-bit platform lists the maximum values for these configuration parameters and platform-specific parameters such as SHMMAX. For more information about the configuration parameters, see the *Administrator's Reference* and the chapter on shared memory in the *Performance Guide*.

Managing Shared Memory

In This Chapter	8-3
Setting Operating-System Shared-Memory Configuration Parameters	8-3
Maximum Shared-Memory Segment Size.	8-4
Using More Than Two Gigabytes of Memory	8-4
Maximum Number of Shared-Memory Identifiers	8-5
Shared-Memory Lower-Boundary Address	8-5
Semaphores	8-6
Setting Database Server Shared-Memory Configuration Parameters	8-6
Setting Parameters for Resident Shared Memory	8-7
Setting Parameters for Virtual Shared Memory	8-8
Setting Parameters for Shared-Memory Performance.	8-9
Setting Shared-Memory Parameters with a Text Editor	8-9
Setting Shared-Memory Parameters with ISA	8-10
Setting Shared-Memory Parameters with ON-Monitor	8-10
Setting SQL Statement Cache Parameters	8-11
Reinitializing Shared Memory	8-12
Turning Residency On or Off for Resident Shared Memory	8-12
Turning Residency On or Off in Online Mode	8-13
Turning Residency On or Off When Restarting the Database Server	8-13
Adding a Segment to the Virtual Portion of Shared Memory	8-14

Monitoring Shared Memory	8-14
Monitoring Shared-Memory Segments	8-14
Monitoring the Shared-Memory Profile and Latches	8-15
Using Command-Line Utilities	8-15
Using IBM Informix Server Administrator	8-16
Using ON-Monitor	8-16
Using SMI Tables	8-16
Monitoring Buffers	8-17
Using Command-Line Utilities	8-17
Using ON-Monitor	8-19
Using SMI Tables	8-20
Monitoring Buffer-Pool Activity	8-20
Using Command-Line Utilities	8-20
Using SMI Tables	8-23

In This Chapter

This chapter tells you how to perform tasks related to managing the use of shared memory with the database server. It assumes you are familiar with the terms and concepts in [Chapter 7, “Shared Memory.”](#)

This chapter describes how to perform the following tasks:

- Set the shared-memory configuration parameters
- Reinitialize shared memory
- Turn residency on or off for the resident portion of the database server shared memory
- Add a segment to the virtual portion of shared memory
- Monitor shared memory

This chapter does not cover the `DS_TOTAL_MEMORY` configuration parameter. This parameter places a ceiling on the allocation of memory for decision-support queries. For information on this parameter, refer to your *Performance Guide*.

Setting Operating-System Shared-Memory Configuration Parameters

Several operating-system configuration parameters can affect the use of shared memory by the database server. Parameter names are not provided because names vary among platforms, and not all parameters exist on all platforms. The following list describes these parameters by function:

- Maximum operating-system shared-memory segment size, expressed in bytes or kilobytes
- Minimum shared-memory segment size, expressed in bytes

UNIX

- Maximum number of shared-memory identifiers
- Lower-boundary address for shared memory
- Maximum number of attached shared-memory segments per process
- Maximum amount of systemwide shared memory
- Maximum number of semaphore identifiers
- Maximum number of semaphores
- Maximum number of semaphores per identifier

On UNIX, the machine notes file contains recommended values that you use to configure operating-system resources. Use these recommended values when you configure the operating system. For information on how to set these operating-system parameters, consult your operating-system manuals.

For specific information about your operating-system environment, refer to the machine notes file that is provided with the database server. For more information about the machine notes file, refer to [“Additional Documentation” on page 13](#) of the Introduction. ♦

Maximum Shared-Memory Segment Size

When the database server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible. The first segment size that the database server tries to acquire is the size of the portion that it is allocating (resident, virtual, or communications), rounded up to the nearest multiple of 8 kilobytes.

The database server receives an error from the operating system if the requested segment size exceeds the maximum size allowed. If the database server receives an error, it divides the requested size by two and tries again. Attempts at acquisition continue until the largest segment size that is a multiple of 8 kilobytes can be created. Then the database server creates as many additional segments as it requires.

Windows

Using More Than Two Gigabytes of Memory

The database server can access shared-memory segments larger than two gigabytes on Windows. However, you must enable this feature with an entry in the Windows boot file.

To add the entry, edit the boot.ini file (located in the top level, or root directory). You can either add a new boot option or use the currently existing boot option. To enable support for more than two gigabytes, add the following text to the end of the boot line:

```
/3GB
```

The following example has support for more than two gigabytes enabled:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows NT
Workstation Version 4.00"
/3GB
```

The maximum size of the shared-memory segment depends on the operating system, but it is approximately 3 gigabytes for Windows without additional drivers.

UNIX

Maximum Number of Shared-Memory Identifiers

Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory. The operating system identifies each shared-memory segment with a shared-memory identifier. For most operating systems, virtual processors receive identifiers on a first-come, first-served basis, up to the limit that is defined for the operating system as a whole. For more information about shared-memory identifiers, refer to [“How Virtual Processors Attach to Shared Memory” on page 7-12](#).

You might be able to calculate the maximum amount of shared memory that the operating system can allocate by multiplying the number of shared-memory identifiers by the maximum shared-memory segment size.

Windows

Shared-Memory Lower-Boundary Address

The default load address for a dynamic link library (DLL) on Windows is 0x10000000. Dynamic Server uses a default shared-memory base address (SHMBASE) of 0x0C000000. Therefore, shared memory in the database server can grow up to 0x10000000, which corresponds to an additional 64 megabytes.

UNIX

If shared memory at startup is more than or close to 64 megabytes, a clash of addresses is possible. In this case, set the SHMBASE parameter in the ONCONFIG file to a higher value; for example 0x20000000. To find the shared-memory size, run **onstat -g seg** and view the total memory.

Semaphores

The database server operation requires one UNIX semaphore for each virtual processor, one for each user who connects to the database server through shared memory (**ipcshm** protocol), six for database server utilities, and sixteen for other purposes.

Setting Database Server Shared-Memory Configuration Parameters

Shared-memory configuration parameters fall into the following categories based on their purposes:

- Parameters that affect the resident portion of shared memory
- Parameters that affect the virtual portion of shared memory
- Parameters that affect performance

You can set shared-memory configuration parameters in the following ways:

- Using a text editor
- IBM Informix Server Administrator (ISA)
- Using ON-Monitor ♦

On UNIX, you must be **root** or user **informix** to use either method. On Windows, you must be a user in the Informix Admin group.

UNIX

Setting Parameters for Resident Shared Memory

Figure 8-1 lists the parameters in the ONCONFIG file that specify the configuration of the buffer pool and the internal tables in the resident portion of shared memory. Before any changes that you make to the configuration parameters take effect, you must shut down and restart the database server. For a description of the configuration parameters, refer to the *Administrator's Reference*.

Figure 8-1
Configuring the Resident Portion of Shared Memory

ONCONFIG Parameter	Purpose
BUFFERS	Specifies the maximum number of shared-memory buffers
LOCKS	Specifies the initial number of locks for database objects; for example, rows, key values, pages, and tables
LOGBUFF	Specifies the size of the logical-log buffers
PHYSBUFF	Specifies the size of the physical-log buffers
RESIDENT	Specifies residency for the resident portion of the database server shared memory
SERVERNUM	Specifies a unique identification number for the database server on the local host computer
SHMTOTAL	Specifies the total amount of memory to be used by the database server

Setting Parameters for Virtual Shared Memory

Figure 8-2 lists the ONCONFIG parameters that you use to configure the virtual portion of shared memory. For more information, see the chapter on configuration effects on memory in your *Performance Guide*.

Figure 8-2
Configuring the Virtual Portion of Shared Memory

ONCONFIG Parameter	Purpose
DS_HASHSIZE	Number of hash buckets for lists in the data-distribution cache.
DS_POOLSIZE	Maximum number of entries in the data-distribution cache.
PC_HASHSIZE	Specifies the number of hash buckets for the UDR cache and other caches that the database server uses. For more information about setting PC_HASHSIZE, refer to your <i>Performance Guide</i> .
PC_POOLSIZE	Specifies the number of UDRs (SPL routines and external routines) that can be stored in the UDR cache. In addition, this parameter specifies the size of other database server caches, such as the typename cache and the opclass cache. For more information about setting PC_POOLSIZE, refer to your <i>Performance Guide</i> .
SHMADD	Specifies the size of dynamically added shared-memory segments
SHMTOTAL	Specifies the total amount of memory to be used by the database server
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory
STACKSIZE	Specifies the stack size for the database server user threads

Setting Parameters for Shared-Memory Performance

Figure 8-3 lists the ONCONFIG parameters that set shared-memory performance options. For more information, see the chapter on configuration parameters in the *Administrator's Reference*.

Figure 8-3
Setting Shared-Memory Performance Options

ONCONFIG Parameter	Purpose
CKPTINTVL	Specifies the maximum number of seconds that can elapse before the database server checks if a checkpoint is needed
CLEANERS	Specifies the number of page-cleaner threads that the database server is to run
LRU_MAX_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to start
LRU_MIN_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to stop
LRUS	Specifies the number of LRU queues for the shared-memory buffer pool
RA_PAGES	Specifies the number of disk pages that the database server should attempt to read ahead when it performs sequential scans of data or index records
RA_THRESHOLD	Specifies the number of unprocessed memory pages that, after they are read, cause the database server to read ahead on disk

Setting Shared-Memory Parameters with a Text Editor

You can use a text editor to set the configuration parameters for resident and virtual shared memory, and shared-memory performance. Locate the parameter in the ONCONFIG file, enter the new value or values, and rewrite the file to disk. Before the changes take effect, however, you must shut down and restart the database server.

Setting Shared-Memory Parameters with ISA

Use ISA to monitor and set the following shared-memory parameters. For more information, see the ISA online help:

- Execute utility commands such as **onmode** and **onstat**.
- Edit ONCONFIG parameters.
- Monitor segments
- Monitor pools
- Monitor resident memory
- Monitor nonresident memory
- Monitor data dictionary cache

UNIX

Setting Shared-Memory Parameters with ON-Monitor

To set the configuration parameters for the resident and virtual portions of shared memory with ON-Monitor, select the **Parameters→Shared-Memory** option.

To determine the page size for your system, choose the **Parameters→Shared-memory** option in ON-Monitor. The database server page size is the last entry on the page.

Important: The configuration parameters *SHMADD* and *SHMTOTAL* affect both the resident and virtual portions of shared memory.

To set the configuration parameters for the following shared-memory performance options with ON-Monitor, select the **Parameters→perFormance** option:

- LRUS
- LRU_MAX_DIRTY
- LRU_MIN_DIRTY
- CKPTINTVL
- RA_PAGES
- RA_THRESHOLD



Setting SQL Statement Cache Parameters

Figure 8-4 shows the different ways that you can configure the SQL statement cache.

Figure 8-4
Configuring the SQL Statement Cache

ONCONFIG Parameter	Purpose	onmode Command
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL STATEMENT CACHE can hold a parsed and optimized SQL STATEMENT.	onmode -e mode
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.	onmode -W STMT_CACHE_HITS
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.	onmode -W STMT_CACHE_NOLIMIT
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.	None
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.	onmode -W STMT_CACHE_SIZE

Use the following **onstat** options to monitor the SQL statement cache:

- **onstat -g ssc** (same as **onstat -g cac stmt**)
- **onstat -g ssc all**
- **onstat -g ssc pool**

For more information on these configuration parameters, **onstat -g** options, and **onmode** commands, see the utilities chapter in the *Administrator's Reference*. For more information on using the SQL statement cache, monitoring it with the **onstat -g** options, and tuning the configuration parameters, see improving query performance in the *Performance Guide*. For details on qualifying and identical statements, see the *IBM Informix Guide to SQL: Syntax*.

Reinitializing Shared Memory

To reinitialize shared memory, take the database server offline and then online. For information on how to take the database server from online mode to offline, refer to [“From Any Mode Immediately to Offline” on page 4-19](#).

Turning Residency On or Off for Resident Shared Memory

You can turn residency on or off for the resident portion of shared memory in either of the following two ways:

- Use the **onmode** utility to reverse the state of shared-memory residency immediately while the database server is in online mode.
- Change the **RESIDENT** parameter in the **ONCONFIG** file to turn shared-memory residency on or off for the next time that you initialize the database server shared memory.

For a description of the resident portion of shared memory, refer to [“Resident Portion of Shared Memory” on page 7-16](#).

Turning Residency On or Off in Online Mode

To turn residency on or off while the database server is in online mode, use the **onmode** utility.

To turn on residency immediately for the resident portion of shared memory, execute the following command:

```
% onmode -r
```

To turn off residency immediately for the resident portion of shared memory, execute the following command:

```
% onmode -n
```

These commands do not change the value of the **RESIDENT** parameter in the **ONCONFIG** file. That is, this change is not permanent, and residency reverts to the state specified by the **RESIDENT** parameter the next time that you initialize shared memory. On UNIX, you must be **root** or user **informix** to turn residency on or off. On Windows, you must be a user in the Informix Admin group to turn residency on or off.

Turning Residency On or Off When Restarting the Database Server

You can use a text editor to turn residency on or off. To change the current state of residency, use a text editor to locate the **RESIDENT** parameter. Set **RESIDENT** to 1 to turn residency on or to 0 to turn residency off, and rewrite the file to disk. Before the changes take effect, you must shut down and restart the database server.

Adding a Segment to the Virtual Portion of Shared Memory

The **-a** option of the **onmode** utility allows you to add a segment of specified size to virtual shared memory.

You do not normally need to add segments to virtual shared memory because the database server automatically adds segments as needed.

The option to add a segment with the **onmode** utility is useful if the number of operating-system segments is limited, and the initial segment size is so low, relative to the amount that is required, that the operating-system limit of shared-memory segments is nearly exceeded.

Monitoring Shared Memory

This section describes how to monitor shared-memory segments, the shared-memory profile, and the use of specific shared-memory resources (buffers, latches, and locks).

You can use the **onstat -o** utility to capture a static snapshot of database server shared memory for later analysis and comparison.

Monitoring Shared-Memory Segments

Monitor the shared-memory segments to determine the number and size of the segments that the database server creates. The database server allocates shared-memory segments dynamically, so these numbers can change. If the database server is allocating too many shared-memory segments, you can increase the SHMVIRTSIZE configuration parameter. For more information, see the chapter on configuration parameters in the *Administrator's Reference*.

The **onstat -g seg** command lists information for each shared-memory segment, including the address and size of the segment, and the amount of memory that is free or in use. [Figure 8-5](#) shows sample output.

```
Segment Summary:
(resident segments are not locked)
id      key      addr      size      ovhd      class blkused  blkfree
300     1381386241 400000    614400    800       R      71         4
301     1381386242 496000    4096000   644       V      322        178
```

Figure 8-5
onstat -g seg Output

Monitoring the Shared-Memory Profile and Latches

Monitor the database server profile to analyze performance and the use of shared-memory resources. The Profile screen maintains cumulative statistics on shared-memory use. To reset these statistics to zero, use the **onstat -z** option. For a description of all the fields that **onstat** displays, see the utilities chapter in the *Administrator's Reference*.

You can obtain statistics on latch use and information on specific latches. These statistics provide a measure of the system activity.

Using Command-Line Utilities

You can use the following command-line utilities to monitor shared memory and latches:

- **onstat -s**
- **onstat -p**

onstat -s

Use **onstat -s** command to obtain latch information.

onstat -p

Execute **onstat -p** to display statistics on database server activity and waiting latches (see the **lchwaits** field). [Figure 8-6](#) shows these statistics.

Profile								
dskreads	pagreads	bufreads	%cached	dskwrits	pagwrits	bufwrits	%cached	
382	400	14438	97.35	381	568	3509	89.14	
isamtot	open	start	read	write	rewrite	delete	commit	rollbk
9463	1078	1584	2316	909	162	27	183	1
ovlock	ovuserthread	ovbuff	usercpu	syscpu	numckpts	flushes		
0	0	0	13.55	13.02	5	18		
bufwaits	lokwaits	lockreqs	deadlks	dltouts	ckpwaits	compress	seqscans	
14	0	16143	0	0	0	101	68	
ixda-RA	idx-RA	da-RA	RA-pgsused	lchwaits				
5	0	204	148	12				

Figure 8-6
onstat -p Output

Using IBM Informix Server Administrator

You can use ISA to obtain information about latches, spin locks, and profiles.

Using ON-Monitor

Select **Status→Profile**. The screen displays shared-memory statistics, as well as the current operating mode, the boot time, the current time, and latches.

Using SMI Tables

Query the **sysprofile** table to obtain shared-memory statistics. This table contains all of the statistics available in **onstat -p** output except the **ovbuff**, **usercpu**, and **syscpu** statistics.

Monitoring Buffers

You can obtain both statistics on buffer use and information on specific buffers.

The statistical information includes the percentage of data writes that are cached to buffers and the number of times that threads had to wait to obtain a buffer. The percentage of writes cached is an important measure of performance. (For information on how to use this statistic to tune the database server, see your *Performance Guide*.) The number of waits for buffers gives a measure of system concurrency.

Information on specific buffers includes a listing of all the buffers in shared memory that are held by a thread. This information allows you to track the status of a particular buffer. For example, you can determine if another thread is waiting for the buffer.

Using Command-Line Utilities

You can use the following command-line utilities to monitor buffers:

- **onstat -p**
- **onstat -B**
- **onstat -b**
- **onstat -X**

onstat -p

Execute **onstat -p** to obtain statistics about cached reads and writes. The following caching statistics appear in four fields on the top row of the output display:

- The number of reads from shared-memory buffers (**bufreads**)
- The percentage of reads cached (**%cached**)
- The number of writes to shared memory (**bufwrits**)
- The percentage of writes cached (**%cached**)
- Information on generic pages (nonstandard pages in the buffer pool)

Figure 8-7 shows these fields.

```
Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
382      400      14438    97.35   381      568      3509    89.14
...
```

Figure 8-7
Cached Read and Write Statistics in the onstat -p Output

The number of reads or writes can appear as a negative number if the number of occurrences exceeds 2^{32} (depends on the platform).

The **onstat -p** option also displays a statistic (**bufwaits**) that indicates the number of times that sessions had to wait for a buffer.

onstat -B

Execute **onstat -B** to obtain the following buffer information:

- Address of every regular shared-memory buffer
- Address of the thread that currently holds the buffer
- Address of the first thread that is waiting for each buffer

Figure 8-8 shows an example of **onstat -B** output.

```
Buffers
address  userthread flgs pagenum  memaddr  nslots  pgflgs xflgs owner  waitlis
t
a0bd068  0          806  1:10264      a0ee800  6       52    0      0
a0bd0e8  0          806  3:24099      a0ef000  1       41    0      0
a0bd168  0           6   1:1          a0ef800  88      15   8067c4  0
...

1 modified, 256 hash buckets, 2048 buffer size
```

Figure 8-8
onstat -B Output

onstat -b

Execute **onstat -b** to obtain the following information about each buffer:

- Address of each buffer currently held by a thread
- Page numbers for the page held in the buffer
- Type of page held in the buffer (for example, data page, tblspace page, and so on)

- Type of lock placed on the buffer (exclusive or shared)
- Address of the thread that is currently holding the buffer
- Address of the first thread that is waiting for each buffer

You can compare the addresses of the user threads to the addresses that appear in the **onstat -u** display to obtain the session ID number. [Figure 8-9](#) shows sample output. For more information on the fields that **onstat** displays, see the utilities chapter of the *Administrator's Reference*.

Buffers										
address	userthread	flgs	pagenum	memaddr	nslots	pgflgs	xflgs	owner	waitlist	
a0f7f28	0	c47	1:14322	a173000	9	11	10	0	0	
a0fa2a8	0	47	1:14310	a196800	0	3	10	0	0	
a0fac28	0	847	1:14307	a1a0000	74	52	10	0	0	
a0fe3a8	0	47	1:14306	a1d7800	0	3	10	0	0	
...										
a133128	0	c47	1:14328	a525000	2	11	10	0	0	
317 modified, 2048 hash buckets, 2048 buffer size										

Figure 8-9
onstat -b Output

onstat -X

Execute **onstat -X** to obtain the same information as for **onstat -b**, along with the *complete* list of all threads that are waiting for buffers, not just the first waiting thread.

UNIX

Using ON-Monitor

To access the fields mentioned on [page 8-17](#) for **onstat -p** (**bufreads**, **%cached**, **bufwrits** **%cached**), select the **Status→Profile** option.

[Figure 8-10](#) shows the output.

...						
Disk Reads	Buf. Reads	%Cached	Disk Writes	Buf. Writes	%Cached	
177	330	46.36	4	0	0.00	
...						

Figure 8-10
Cached Read and Write Statistics in the Profile Option of the ON-Monitor Status Menu

Using SMI Tables

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

Row	Description
dskreads	Number of reads from disk
bufreads	Number of reads from buffers
dskwrites	Number of writes to disk
bufwrites	Number of writes to buffers
buffwts	Number of times that any thread had to wait for a buffer

Monitoring Buffer-Pool Activity

You can obtain statistics that relate to buffer availability as well as information on the buffers in each LRU queue.

The statistical information includes the number of times that the database server attempted to exceed the maximum number of buffers and the number of writes to disk (categorized by the event that caused the buffers to flush). These statistics help you determine if the number of buffers is appropriate. For information on tuning database server buffers, see your *Performance Guide*.

Information on the buffers in each LRU queue consists of the length of the queue and the percentage of the buffers in the queue that have been modified.

Using Command-Line Utilities

You can use the **onstat** utility to obtain information on buffer-pool activity. You also can execute **onstat** options from ISA.

For more information about the **onstat** options, refer to the utilities chapter of the *Administrator's Reference*.

onstat -p

The **onstat -p** output contains a statistic (**ovbuff**) that indicates the number of times the database server attempted to exceed the maximum number of shared buffers specified by the **BUFFERS** parameter in the **ONCONFIG** file.

[Figure 8-11](#) shows **onstat -p** output, including the **ovbuff** field.

```
...
ovtbls   ovlock   ovuserthread  ovbuff   usercpu   syscpu   numckpts  flushes
0         0         0           0       13.55     13.02     5         18
...
```

Figure 8-11
onstat -p Output
Showing **ovbuff**
Field

onstat -F

Execute **onstat -F** to obtain a count by write type of the writes performed. (For an explanation of the different write types, see [“Describing Flushing Activity” on page 7-43](#).) [Figure 8-12 on page 8-22](#) shows an example of the output. This information tells you when and how the buffers are flushed.

The **onstat -F** command displays totals for the following write types:

- Foreground write
- LRU write
- Chunk write

The **onstat -F** command also lists the following information about the page cleaners:

- Page-cleaner number
- Page-cleaner shared-memory address
- Current state of the page cleaner
- LRU queue to which the page cleaner was assigned

Figure 8-12 shows an example of the **onstat -F** output.

```
...

Fg Writes      LRU Writes      Chunk Writes
0              146             140

address flusher state  data
8067c4    0      I      0      = 0X0
      states: Exit Idle Chunk Lru
```

Figure 8-12
onstat -F Output

onstat -R

Execute **onstat -R** to obtain information about the number of buffers in each LRU queue and the number and percentage of the buffers that are modified or free. Figure 8-13 shows an example of **onstat -R** output.

```
8 buffer LRU queue pairs
# f/m length % of pair total
0 f      3    37.5%      8
1 m      5    55.6%
2 f      5    45.5%     11
3 m      6    54.5%
4 f      2    18.2%     11
5 m      9    81.8%
6 f      5    50.0%     10
7 m      5    55.6%
8 F      5    50.0%     10
9 m      5    45.5%
10 f     0     0.0%     10
11 m     10   100.0%
12 f      1    11.1%      9
13 m      8    88.9%
14 f      2    28.6%      7
15 m      5    71.4%

53 dirty, 76 queued, 80 total, 128 hash buckets, 2048 buffer size
start clean at 60% (of pair total) dirty, or 6 buffs dirty, stop at 50%
```

Figure 8-13
onstat -R Output

Using SMI Tables

Query the **sysprofile** table to obtain the statistics on write types that are held in the following rows.

Row	Description
fgwrites	Number of foreground writes
lrwwrites	Number of LRU writes
chunkwrites	Number of chunk writes

Data Storage

In This Chapter	9-5
Physical and Logical Units of Storage	9-5
Chunks	9-6
Disk Allocation for Chunks	9-7
Disk Access on Windows	9-7
Unbuffered or Buffered Disk Access on UNIX	9-8
Offsets	9-9
Pages	9-10
Blobpages	9-11
Sbpages.	9-12
Extents	9-14
Dbspaces	9-16
Control of Where Data Is Stored	9-16
Root Dbspace	9-18
Temporary Dbspaces	9-19
Blobspaces.	9-20
Sbspaces	9-21
Advantages of Using Sbspaces	9-21
Sbspaces and Enterprise Replication	9-22
Metadata, User Data, and Reserved Area	9-22
Control of Where Data Is Stored	9-23
Storage Characteristics of Sbspaces	9-25
Extent Sizes for Sbspaces	9-25
Average Smart-Large-Object Size	9-26

Buffering Mode	9-26
Last-Access Time	9-27
Lock Mode	9-27
Logging	9-27
Levels of Inheritance for Sbspace Characteristics	9-28
More Information About Sbspaces	9-30
Temporary Sbspaces	9-32
Comparison of Temporary and Standard Sbspaces.	9-33
Temporary Smart Large Objects	9-34
Extspaces	9-35
Databases	9-35
Tables	9-36
Table Types for Dynamic Server	9-38
Standard Permanent Tables	9-39
RAW Tables	9-39
Temp Tables	9-40
Properties of Table Types	9-40
Loading of Data Into a Table	9-40
Fast Recovery of Table Types	9-41
Backup and Restore of RAW Tables	9-41
Temporary Tables	9-42
Temporary Tables That You Create	9-43
Temporary Tables That the Database Server Creates	9-44
Tblspaces	9-45
Maximum Number of Tblspaces in a Table	9-46
Table and Index Tblspaces	9-46
Extent Interleaving	9-48
Table Fragmentation and Data Storage	9-49
Amount of Disk Space Needed to Store Data	9-49
Size of the Root Dbspace.	9-49
Physical and Logical Logs	9-50
Temporary Tables.	9-51
Critical Data	9-51
Extra Space	9-51
Amount of Space That Databases Require.	9-52

Disk-Layout Guidelines	9-52
Dbospace and Chunk Guidelines.	9-53
Table-Location Guidelines.	9-54
Sample Disk Layouts	9-55
Sample Layout When Performance Is Highest Priority	9-56
Sample Layout When Availability Is Highest Priority	9-58
Logical-Volume Manager	9-60

In This Chapter

This chapter defines terms and explains the concepts that you must understand to perform the tasks described in [Chapter 10, “Managing Disk Space.”](#) This chapter covers the following topics:

- Definitions of the physical and logical units that the database server uses to store data on disk
- Instructions on how to calculate the amount of disk space that you need to store your data
- Guidelines on how to lay out your disk space and where to place your databases and tables

The release notes file contains supplementary information on the maximum values related to the storage units discussed in this chapter. For information on how to access this file, see [“Additional Documentation” on page 13](#) of the Introduction.

Physical and Logical Units of Storage

The database server uses the physical units of storage to allocate disk space. Unlike the logical units of storage whose size fluctuates, each of the physical units has a fixed or assigned size that is determined by the disk architecture. The database server uses the following physical units to manage disk space:

- Chunk
- Page
- Extent
- Blobpage
- Sbpage

The database server stores data in the following logical units:

- Dbspace
- Temporary dbspace
- Blobspace
- Sbspace
- Temporary sbspace
- Extspace
- Database
- Table
- Tblspace

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

- Logical log
- Physical log
- Reserved pages

The following sections describe the various data-storage units that the database server supports and the relationships between those units. For information about reserved pages, see the disk structures and storage chapter in the *Administrator's Reference*.

Chunks

A *chunk* is the largest unit of physical disk dedicated to database server data storage. Chunks provide administrators with a significantly large unit for allocating disk space. The maximum size of an individual chunk is 4 terabytes. The number of allowable chunks is 32,766. You must run onmode -BC to enable the maximum size of a chunk and the maximum number allowable. If onmode -BC is not run, then the maximum chunk size is 2 gigabytes.

The database server administrator adds one or more chunks to the following storage spaces when they approach full capacity:

- Dbspace (see [page 9-16](#))
- Blobospace (see [page 9-20](#))
- Sbspace (see [page 9-21](#))

For information on chunk names, size, and number, see “[Specifying Names for Storage Spaces and Chunks](#)” on page 10-12 and “[Specifying the Maximum Size of Chunks](#)” on page 10-13.

The database server also uses chunks for mirroring. A *primary chunk* is a chunk from which the database server copies data to a *mirrored chunk*. If the primary chunk fails, the database server brings the mirrored chunk online automatically. For more information, see [Chapter 17, “Mirroring.”](#)

Disk Allocation for Chunks

The database server can use regular operating-system files or *raw disk devices* to store data. On UNIX, it is recommended that you use raw disk devices to store data whenever performance is important. On Windows, using NTFS files to store data is recommended for ease of administration.

An Informix storage space can reside on an NFS-mounted directory only if the vendor of that NFS device is certified by IBM. For information about NFS products that you can use to mount a storage space for an Informix database server, see the product compatibility information on the IBM Informix web site at <http://www.ibm.com/software/data/informix/pubs/smv/index.html>.

Windows

Disk Access on Windows

On Windows, both raw disks and NTFS use kernel asynchronous I/O (KAIO). The Windows file system manager adds additional overhead to disk I/O, so using raw disks provides slight performance advantages. Because NTFS files are a more standard method of storing data, it is recommended you use NTFS files instead of raw disks. Consider using raw disks if your database server requires a large amount of disk access.

UNIX

Raw Disk Space on Windows

On Windows, *raw disk space* can be either a physical drive without a drive letter or a logical disk partition that has been assigned a drive letter using the **Disk Administrator**. The space can either be formatted or unformatted. If it contains data, the data will be overwritten after the space has been allocated to the database server. For more information, see [“Allocating Raw Disk Space on Windows” on page 10-11](#).

NTFS Files

You must use NTFS files, not FAT files, for disk space on Windows. For more information, see [“Allocating NTFS File Space on Windows” on page 10-10](#).

Unbuffered or Buffered Disk Access on UNIX

You can allocate disk space in two ways:

- Use files that are buffered through the operating system, also referred to as *cooked* files.
- Use unbuffered disk access, also called *raw* disk space.

Raw Disk Space on UNIX

When dbspaces reside on *raw disk devices* (also called *character-special devices*), the database server uses unbuffered disk access. Performance is much better when you use raw disk devices than cooked files because the database server has direct I/O access to the devices. A raw disk directly transfers data between the database server memory and disk without also copying data.

To create a raw device, configure a *block device* (hard disk) with a raw interface. The storage space that the device provides is called *raw disk space*. A chunk of raw disk space is physically contiguous.

The name of the chunk is the name of the character-special file in the **/dev** directory. In many operating systems, you can distinguish the character-special file from the block-special file by the first letter in the filename (typically *r*). For example, **/dev/r***sd0f* is the character-special device that corresponds to the **/dev/sd0f** block-special device.

For more information, see [“Allocating Raw Disk Space on UNIX” on page 10-9](#).

Cooked Files

A cooked file is a regular file that the operating system manages. Cooked file chunks and raw disk chunks are equally reliable. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

You can more easily allocate cooked files than raw disk space. To allocate a cooked file, you need only create the file on any existing partition. The name of the chunk is the complete pathname of the file. These steps are described in detail in [“Allocating Cooked File Spaces on UNIX” on page 10-8](#).

In a learning environment, where performance is not critical, or for static data, cooked files can be convenient. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system with minimal activity.



Warning: *Chunks located in file systems perform worse than chunks located on raw disk due to operating-system overhead. For cooked file chunks, the operating system processes all chunk I/O from its own buffer pool and ensures that all writes to chunks are physically written to the disk.*

Offsets

The system administrator might divide a physical disk into *partitions*, which are different parts of a disk that have separate pathnames. Although it is recommended that you use an entire disk partition when you allocate a chunk on a raw disk device, you can subdivide partitions or cooked files into smaller chunks using *offsets*. For more information, see [“Disk-Layout Guidelines” on page 9-52](#).



Tip: *With a 4-terabyte limit to the size of a chunk, you can avoid partitioning a disk by assigning a single chunk per disk drive.*

An offset allows you to indicate the location of a given chunk on the disk partition, file, or device. For example, suppose that you create a 1000-kilobyte chunk that you want to divide into two chunks of 500 kilobytes each. You can use an offset of zero kilobytes to mark the beginning of the first chunk and an offset of 500 kilobytes to mark the beginning of the second chunk.

You can specify an offset whenever you create, add, or drop a chunk from a dbspace, blobspace, or sbspace.

You might also need to specify an offset to prevent the database server from overwriting partition information. [“Allocating Raw Disk Space on UNIX” on page 10-9](#) explains when and how to specify an offset.

Pages

A *page* is the physical unit of disk storage that the database server uses to read from and write to Informix databases. [Figure 9-1](#) illustrates the concept of a page, represented by a darkened sector of a disk platter.

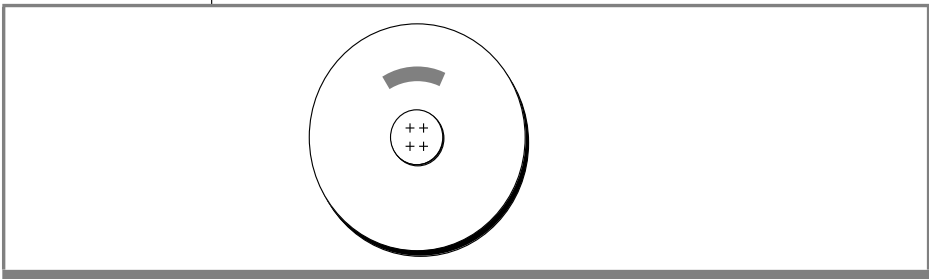


Figure 9-1
A Page on Disk

On most UNIX platforms, the page size is 2 kilobytes. On Windows, the page size is 4 kilobytes. Because your hardware determines the size of your page, you cannot alter this value.

A chunk contains a certain number of pages, as [Figure 9-2](#) illustrates. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

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

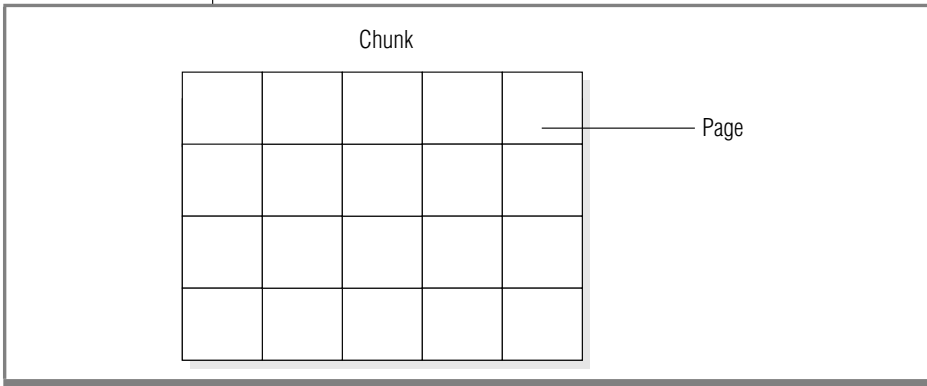


Figure 9-2
A Chunk, Logically Separated into a Series of Pages

Blobspages

A blobpage is the unit of disk-space allocation that the database server uses to store simple large objects (TEXT or BYTE data) within a blobspace. For a description of blobspaces, refer to [“Blobspaces” on page 9-20](#).

You specify blobpage size as a multiple of the database server page size. Because the database server allocates blobpages as contiguous spaces, it is more efficient to store simple large objects in blobpages that are as close to the size of the data as possible. [Figure 9-3](#) illustrates the concept of a blobpage, represented as a multiple (three) of a data page.

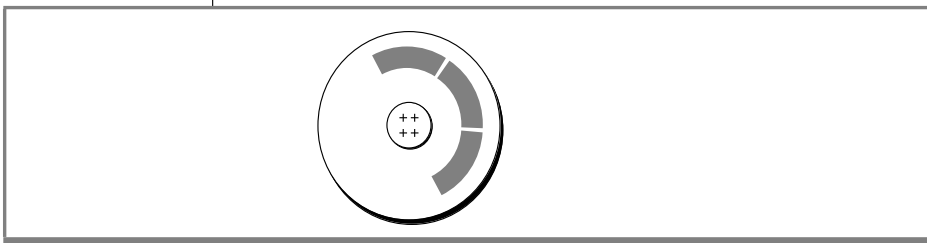


Figure 9-3
A Blobpage on Disk

For information about how Dynamic Server structures data stored in a blobpage, see structure of a blobspace blobpage in the disk structures and storage chapter of the *Administrator's Reference*.

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as [Figure 9-4](#) illustrates. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

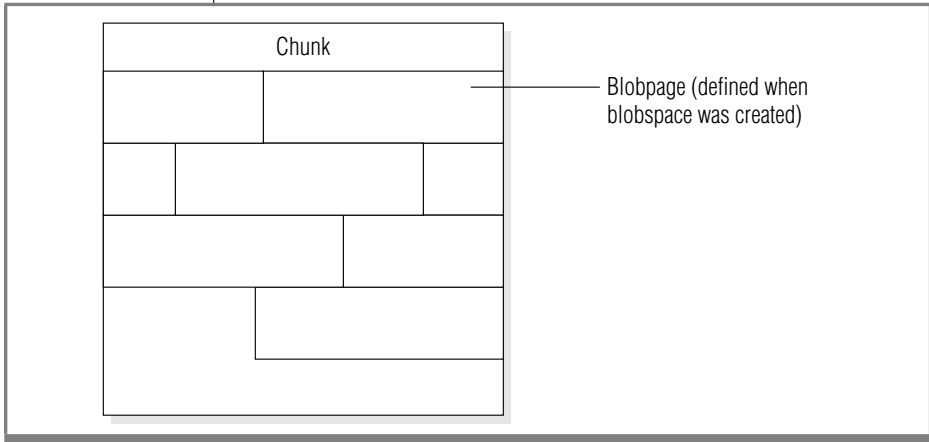


Figure 9-4
A Chunk in a
Blobspace, Logically
Separated into a
Series of Blobpages

Instead of storing simple-large-object data in a blobpage, you can choose to store it in a dbspace. However, for a simple large object larger than two pages, performance improves when you store it in a blobspace. Simple large objects stored in a dbspace can share a page, but simple large objects stored in a blobspace do not share pages.

For information on how to determine the size of a blobpage, refer to [“Determining Blobpage Size” on page 10-21](#).

Sbpages

An sbpage is the type of page that the database server uses to store smart large objects within a sbpace. For a description of sbspaces, refer to [“Sbspaces” on page 9-21](#). Unlike blobpages, sbpages are not configurable. An sbpage is the same size as the database server page, which is usually 2 kilobytes on UNIX and 4 kilobytes on Windows.

The unit of allocation in an sbospace is an extent, whereas the unit of allocation in a blobspace is a blobpage. Just as with pages in a chunk, a certain number of smart large object extents compose a chunk in an sbospace, as [Figure 9-5](#) illustrates. An extent is always entirely contained in a chunk and cannot cross chunk boundaries.

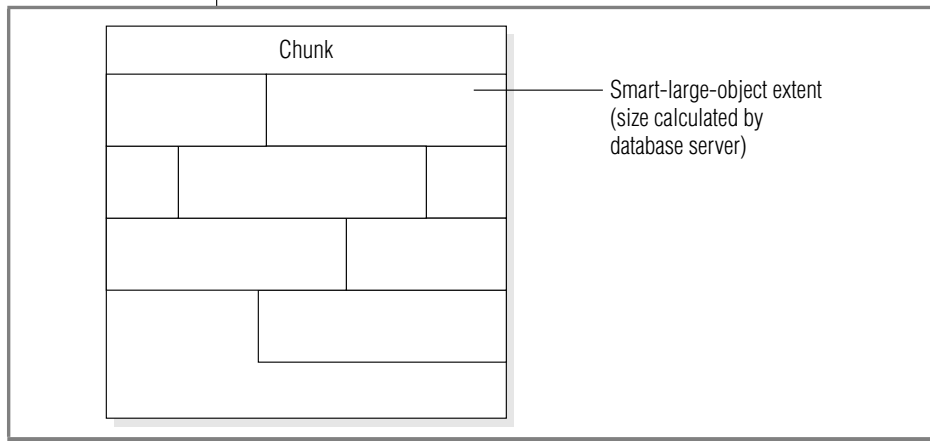


Figure 9-5
A Chunk in an
Sbospace, Logically
Separated into a
Series of Extents

Smart large objects cannot be stored in a dbospace or blobspace. For more information, see [“Sbspaces” on page 9-21](#), and sbospace structure in the disk structures and storage chapter of the *Administrator’s Reference*.

The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For more information, see [“Extent Sizes for Sbspaces” on page 9-25](#).

Extents

When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. When this space fills, the database server must allocate space for additional storage. The physical unit of storage that the database server uses to allocate both the initial and subsequent storage space is called an *extent*. [Figure 9-6](#) illustrates the concept of an extent.

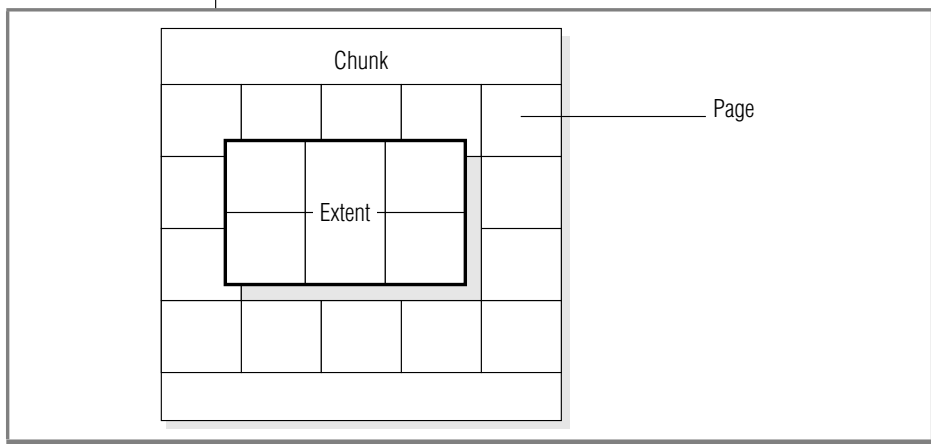


Figure 9-6
An Extent That
Consists of Six
Contiguous Pages
on a Raw Disk
Device

An extent consists of a collection of contiguous pages that store data for a given table. (See [“Tables” on page 9-36](#).) Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of kilobytes allocated to the table when it is first created. The *next-extent* size is the number of kilobytes allocated to the table when the initial extent (and any subsequent extents) becomes full. For permanent tables and user-defined temporary tables, the next-extent size begins to double after 16 extents have been added. For system-created temporary tables, the next-extent size begins to double after 4 extents have been added.

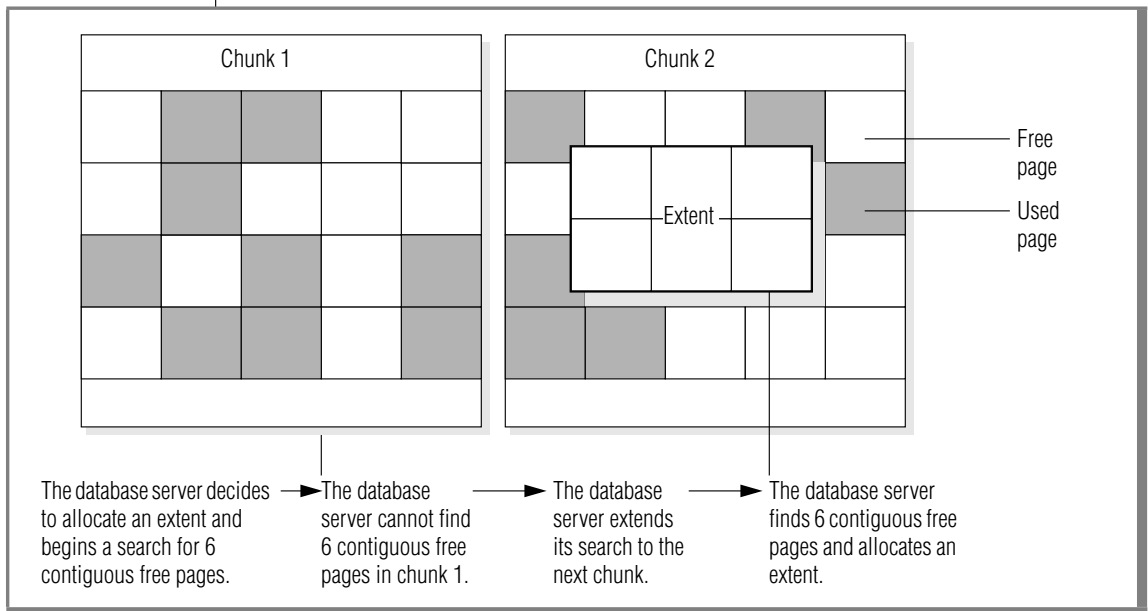
To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the *IBM Informix Guide to SQL: Syntax* and disk structures in the *IBM Informix Dynamic Server Administrator's Reference*.

When you create a table with a column for CLOB or BLOB data types, you also define extents for an sbspace. For more information, refer to [“Storage Characteristics of Sbspaces” on page 9-25](#).

Figure 9-7 shows how the database server allocates six pages for an extent:

- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.
- If the database server cannot find the contiguous disk space that is specified for the next-extent size, it searches the next chunk in the dbspace for contiguous space.

Figure 9-7
Process of Extent Allocation



Dbspaces

A *dbspace* is a logical unit that can contain between 1 and 32,767 chunks. Place databases, tables, logical-log files, and the physical log in dbspaces.

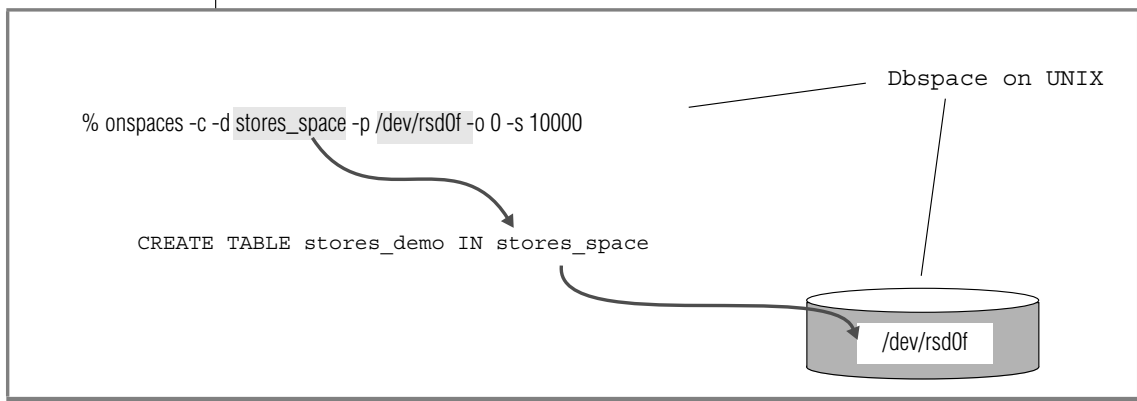
Control of Where Data Is Stored

A key responsibility of the database server administrator is to control where the database server stores data. By storing high-use access tables or *critical dbspaces* (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on separate physical devices, you ensure that when one of the disks holding noncritical data fails, the failure affects only the availability of data on that disk.

As [Figure 9-8](#) shows, to control the placement of databases or tables, you can use the *IN dbspace* option of the CREATE DATABASE or CREATE TABLE statements. (See [“Tables”](#) on page 9-36.)

Figure 9-8

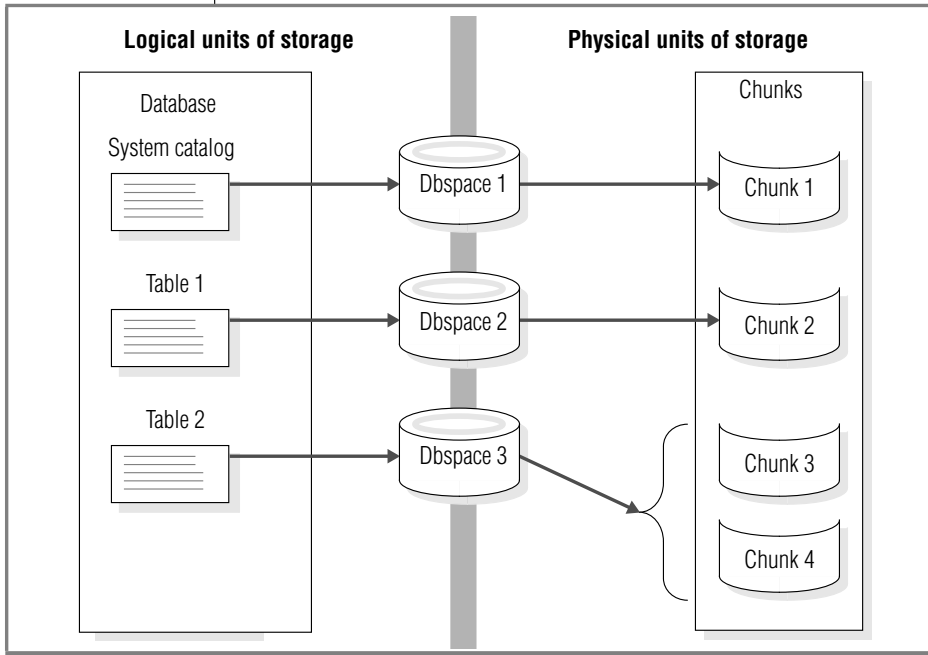
Controlling Table Placement with the CREATE TABLE... IN Statement



Before you create a database or table in a dbspace, you must first create the dbspace. For more information on how to create a dbspace, see [“Creating a Dbospace”](#) on page 10-14.

A dbspace includes one or more chunks, as [Figure 9-9](#) shows. You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor dbspace chunks for fullness and to anticipate the need to allocate more chunks to a dbspace. (See [“Monitoring Disk Usage” on page 10-41.](#)) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

Figure 9-9
*Dbspaces That Link
Logical and Physical
Units of Storage*



The database server uses the dbspace to store databases and tables. (See [“Tables” on page 9-36.](#))

You can mirror every chunk in a mirrored dbspace. As soon as the database server allocates a mirrored chunk, it flags all space in that mirrored chunk as full. See [“Monitoring Disk Usage” on page 10-41.](#)

For information on using ISA or **onspaces** to perform the following tasks, see [Chapter 10, “Managing Disk Space.”](#)

- Creating a dbspace
- Adding a chunk to a dbspace

- Dropping a chunk
- Dropping a dbspace, blobspace, or sbspace

Root Dbspace

The root dbspace is the initial dbspace that the database server creates. The root dbspace is special because it contains reserved pages and internal tables that describe and track all physical and logical units of storage. (For more information on these topics, see [“Tables” on page 9-36](#) and the disk structures and storage chapter in the *Administrator’s Reference*.) The initial chunk of the root dbspace and its mirror are the only chunks created during disk-space initialization. You can add other chunks to the root dbspace after disk-space initialization.

The following disk-configuration parameters in the ONCONFIG configuration file refer to the first (initial) chunk of the root dbspace:

- ROOTPATH
- ROOTOFFSET
- ROOTNAME
- MIRRORPATH
- MIRROROFFSET

The root dbspace is also the default dbspace location for any database created with the CREATE DATABASE statement.

The root dbspace is the default location for all temporary tables created by the database server to perform requested data management.

[“Size of the Root Dbspace” on page 9-49](#) explains how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you initialize database server disk space.

Temporary Dbspaces

A temporary dbspace is a dbspace reserved exclusively for the storage of temporary tables. You cannot mirror a temporary dbspace.

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

Whenever you initialize the database server, all temporary dbspaces are reinitialized. The database server clears any tables that might be left over from the last time that the database server shut down.

The database server does not perform logical or physical logging for temporary dbspaces. Because temporary dbspaces are not physically logged, fewer checkpoints and I/O operations occur, which improves performance.

The database server logs table creation, the allocation of extents, and the dropping of the table for a temporary table in a standard dbspace. In contrast, the database server does not log tables stored in temporary dbspaces. Logical-log suppression in temporary dbspaces reduces the number of log records to roll forward during logical recovery as well, thus improving the performance during critical down time.

Using temporary dbspaces to store temporary tables also reduces the size of your storage-space backup, because the database server does not back up temporary dbspaces.

If you have more than one temporary dbspace and execute a SELECT statement into a temporary table, the results of the query are inserted in round robin order.

For detailed instructions on how to create a temporary dbspace, see [“Creating a Temporary Dbpace” on page 10-16](#).



Important: When the database server is running as a secondary database server in an HDR pair, it requires a temporary dbspace to store any internal temporary tables generated by read-only queries.

Blobspaces

A blobspace is a logical storage unit composed of one or more chunks that store only TEXT and BYTE data. A blobspace stores TEXT and BYTE data in the most efficient way possible. You can store TEXT and BYTE columns associated with distinct tables (see [“Tables” on page 9-36](#)) in the same blobspace.

The database server writes data stored in a blobspace directly to disk. This data does not pass through resident shared memory. If it did, the volume of data could occupy so many of the buffer-pool pages that other data and index pages would be forced out. For the same reason, the database server does not write TEXT or BYTE objects that are assigned to a blobspace to either the logical or physical log. The database server logs blobspace objects by writing them directly from disk to the logical-log backup tapes when you back up the logical logs. Blobspace objects never pass through the logical-log files.

When you create a blobspace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of a database server administrator is to monitor the chunks for fullness and anticipate the need to allocate more chunks to a blobspace. For instructions on how to monitor chunks for fullness, see [“Monitoring Simple Large Objects in a Blobspace” on page 10-49](#). For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see [Chapter 10, “Managing Disk Space.”](#)

For information about the structure of a blobspace, see the chapter on disk structures and storage in the *Administrator’s Reference*.

Sbspaces

An sbspace is a logical storage unit composed of one or more chunks that store *smart large objects*. Smart large objects consist of CLOB (character large object) and BLOB (binary large object) data types. User-defined data types can also use sbspaces. For more information about data types, refer to the *IBM Informix Guide to SQL: Reference*.

Advantages of Using Sbspaces

Sbspaces have the following advantages over blobspaces:

- They have read, write, and seek properties similar to a standard UNIX file.
Programmers can use functions similar to UNIX and Windows functions to read, write, and seek smart large objects. Dynamic Server provides this smart-large-object interface in the DataBlade API and the ESQL/C programming interface.
- They are recoverable.
You can log all write operations on data stored in sbspaces. You can commit or rollback changes if a failure occurs during a transaction.
- They obey transaction isolation modes.
You can lock smart large objects at different levels of granularity, and the lock durations obey the rules for transaction isolation levels. For more information on locking and concurrency, refer to your *Performance Guide*.
- Smart large objects within table rows do not need to be retrieved in one statement.
An application can store or retrieve smart large objects in pieces using either the DataBlade API or the ESQL/C programming interface. For more information on the DataBlade API functions, refer to the *IBM Informix DataBlade API Function Reference*. For more information on the ESQL/C functions, refer to the *IBM Informix ESQL/C Programmer's Manual*.

Sbspaces and Enterprise Replication

Before you define a replication server for Enterprise Replication, you must create an sbspace. Enterprise Replication spools the replicated data to smart large objects. Specify the sbspace name in the CDR_QDATA_SBSPACE configuration parameter. Enterprise Replication uses the default log mode with which the sbspace was created for spooling the row data. The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Metadata, User Data, and Reserved Area

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

- *Metadata area*

Metadata identifies key aspects of the sbspace and each smart large object stored in the sbspace, and enables the database server to manipulate and recover smart large objects stored within.

- *User-data area*

User data is the smart large object data stored in the sbspace by user applications. The chunk has up to two user-data areas.

- *Reserved area*

The database server allocates space from the reserved area to either the metadata or user-data area when more space is needed. The chunk has up to two reserved areas.

For information on correctly allocating metadata and user data for sbspaces, see [“Sizing Sbspace Metadata” on page 10-25](#) and the *Performance Guide*.

When you add a chunk to an sbspace, you can specify whether it contains a metadata area and user-data area or whether to reserve the chunk exclusively for user data. You can add more chunks at any time. If you are updating smart large objects, I/O to the user data is much faster on raw disks than cooked chunk files. For instructions on how to create a sbspace, add chunks to a sbspace, or drop a chunk from a sbspace, see [Chapter 10, “Managing Disk Space.”](#)

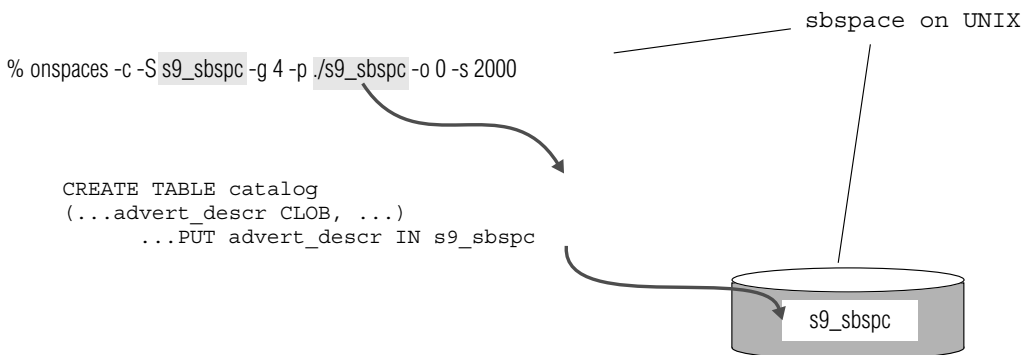


Important: *Sbspace metadata is always logged, regardless of the logging setting of the database.*

Control of Where Data Is Stored

You specify the data type of a column when you create the table. For smart large objects, you specify CLOB, BLOB, or user-defined data types. As [Figure 9-10](#) shows, to control the placement of smart large objects, you can use the IN *sbspace* option in the PUT clause of the CREATE TABLE statement.

Figure 9-10
Controlling Smart-Large-Object Placement



Before you specify an sbspace in a PUT clause, you must first create the sbspace. For more information on how to create an sbspace with the **onspaces -c -S** utility, see [“Adding a Chunk to a Dbspace or Blobspace” on page 10-17](#). For more information on how to specify smart large object characteristics in the PUT clause, refer to the CREATE TABLE statement in the *IBM Informix Guide to SQL: Syntax*.

If you do not specify the PUT clause, the database server stores the smart large objects in the default sbspace that you specify in the SBSPACENAME configuration parameter. For more information on SBSPACENAME, refer to the configuration parameter chapter of the *Administrator's Reference*.

An sbspace includes one or more chunks, as [Figure 9-11](#) shows. When an sbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor sbspace chunks for fullness and to anticipate the need to allocate more chunks to a sbspace. For more information on monitoring sbspaces, refer to your *Performance Guide*.

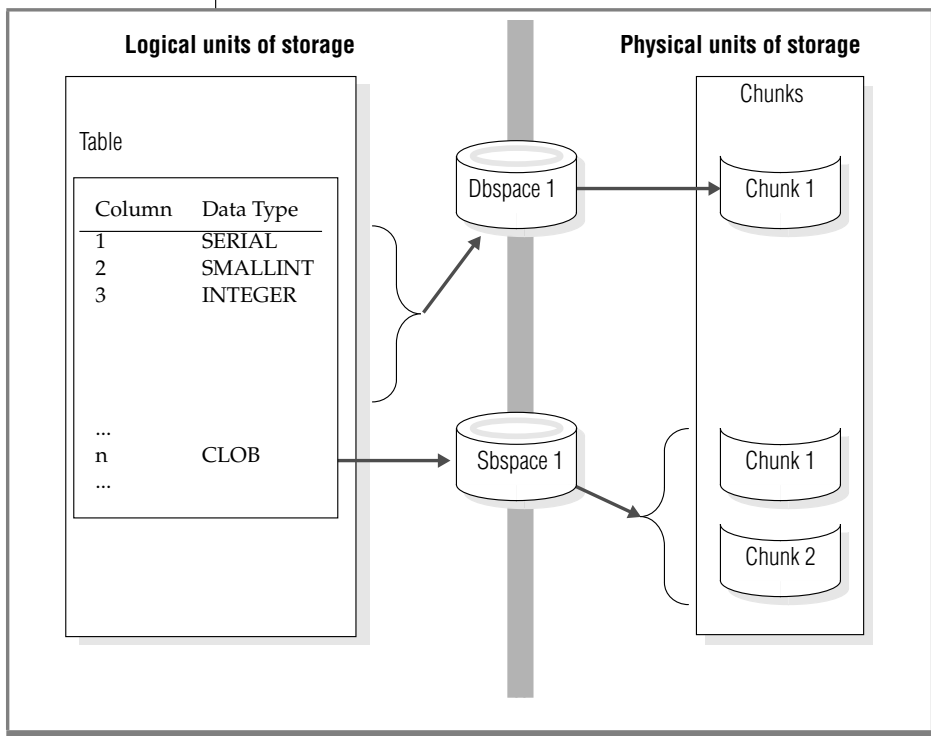


Figure 9-11
*Sbspaces That Link
Logical and Physical
Units of Storage*

The database server uses sbspaces to store table columns that contain smart large objects. The database server uses dbspaces to store the rest of the table columns.

You can mirror an sbspace to speed recovery in event of a media failure. For more information, refer to [“Mirroring” on page 17-3](#).

For information on using **onspaces** to perform the following tasks, see [Chapter 10, “Managing Disk Space.”](#)

- Creating an sbspace
- Adding a chunk to an sbspace
- Altering storage characteristics of smart large objects
- Creating a temporary sbspace
- Dropping an sbspace

Storage Characteristics of Sbspaces

As the database server administrator, you can use the system default values for these storage characteristics, or you can specify them in the **-Df** tags when you create the sbspace with **onspaces -c**. Later on, you can change these sbspace characteristics with the **onspaces -ch** option. The administrator or programmer can override these default values for storage characteristics and attributes for individual tables.

Extent Sizes for Sbspaces

Similar to extents in a table, an extent in an sbspace consists of a collection of contiguous pages that store smart large object data.

The unit of allocation in an sbspace is an extent. The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For example, if an operation asks to write 30 kilobytes, the database server tries to allocate an extent the size of 30 kilobytes.

Important: For most applications, it is recommended that you use the values that the database server calculates for the extent size.



If you know the size of the smart large object, you can use one of the following functions to set the extent size. The database server allocates the entire smart large object as one extent (if an extent of that size is available in the chunk):

- The DataBlade API **mi_lo_specset_estbytes()** function
For more information on the DataBlade API functions for smart large objects, refer to the *IBM Informix DataBlade API Function Reference*.
- The ESQL/C **ifx_lo_specset_estbytes** function
For more information on the ESQL/C functions for smart large objects, refer to the *IBM Informix ESQL/C Programmer's Manual*.

For information about tuning extent sizes, see smart large objects in the chapter on configuration effects on I/O utilization in your *Performance Guide*.

Average Smart-Large-Object Size

Smart large objects usually vary in length. You can provide an average size of your smart large objects to calculate space for an sbspace. You specify this average size with the **AVG_LO_SIZE** tag of the **onspaces -c -Df** option.

To specify the size and location of the metadata area, specify the **-Ms** and **-Mo** flags in the **onspaces** command. If you do not use the **-Ms** flag, the database server uses the value of **AVG_LO_SIZE** to estimate the amount of space to allocate for the metadata area. For more information, refer to [“Sizing Sbspace Metadata” on page 10-25](#).

Buffering Mode

When you create an sbspace, the default buffering mode is on, which means to use the buffer pool in the resident portion of shared memory.

As the database administrator, you can specify the buffering mode with the **BUFFERING** tag of the **onspaces -c -Df** option. The default is **“BUFFERING=ON”**, which means to use the buffer pool. If you turn off buffering, the database server uses private buffers in the virtual portion of shared memory.



Important: In general, if read and write operations to the smart large objects are less than 8 kilobytes, do not specify a buffering mode when you create the sbspace. If you are reading or writing short blocks of data, such as 2 kilobytes or 4 kilobytes, leave the default of “BUFFERING=ON” to obtain better performance.

For information about when to use private buffers, see the section on light-weight I/O operations in the chapter on configuration effects on I/O utilization in your *Performance Guide*.

Last-Access Time

When you create an sbspace, you can specify whether or not the database server should keep the last time that the smart large object was read or updated with the ACCESTIME tag of the **onspaces -c -Df** option. The default is “ACCESTIME=OFF”. The database server keeps this last-access time in the metadata area.

For more information on how programmers use this last-access time, refer to the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Lock Mode

When you create an sbspace, you can specify whether or not the database server should lock the whole smart large object or a range of bytes within a smart large object with the LOCK_MODE tag of the **onspaces -c -Df** option. The default is “LOCK_MODE=BLOB”, which means to lock the entire smart large object. For more information, refer to the locking chapter in your *Performance Guide*.

Logging

When you create an sbspace, you can specify whether or not to turn on logging for the smart large objects. The default is no logging. For more information, refer to [“Logging Sbspaces and Smart Large Objects” on page 13-13](#).



Important: When you use logging databases, turn logging on for the sbspaces. If a failure occurs that requires log recovery, you can keep the smart large objects consistent with the rest of the database.

You specify the logging status with the LOGGING tag of the **onspaces -c -Df** option. The default is "LOGGING=OFF". You can change the logging status with the **onspaces -ch -Df** option. You can override this logging status with the PUT clause in the SQL statements CREATE TABLE or ALTER TABLE. For more information about these SQL statements, refer to the *IBM Informix Guide to SQL: Syntax*.

The programmer can override this logging status with functions that the DataBlade API and ESQL/C provide. For more information on the DataBlade API functions for smart large objects, refer to the *IBM Informix DataBlade API Function Reference*. For more information on the ESQL/C functions for smart large objects, refer to the *IBM Informix ESQL/C Programmer's Manual*.

When you turn on logging for an sbspace, the smart large objects pass through the resident portion of shared memory. Although applications can retrieve pieces of a smart large object, you still need to consider the larger size of data that might pass through the buffer pool and logical-log buffers. For more information, refer to ["Accessing Smart Large Objects" on page 7-49](#).

Levels of Inheritance for Sbspace Characteristics

The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. You can use the system default values for sbspace attributes, or override them for specific sbspaces, columns in a table, or smart large objects. [Figure 9-12](#) shows the storage-characteristics hierarchy for a smart large object.

Figure 9-12
*Storage-
Characteristics
Hierarchy*

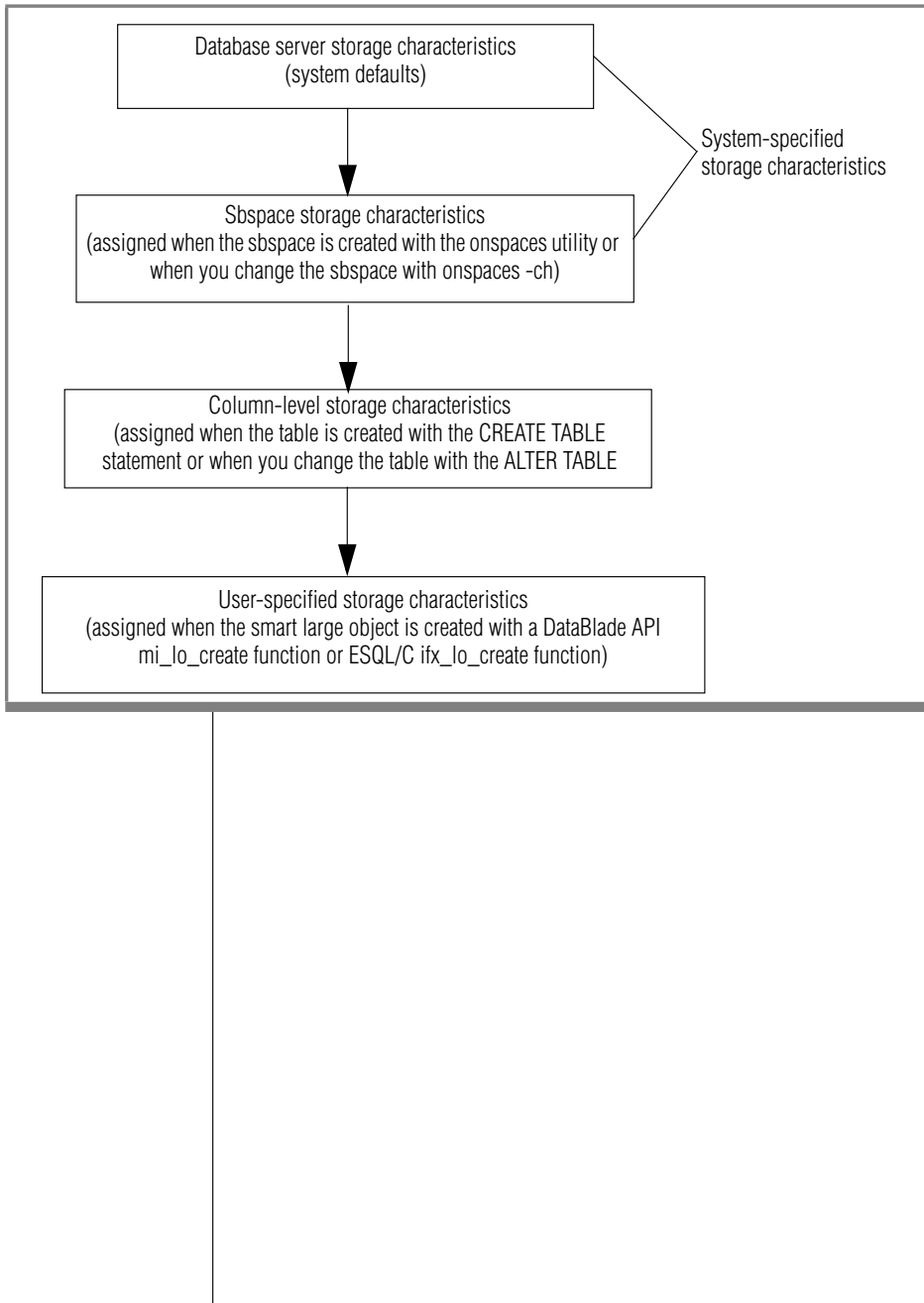


Figure 9-12 shows that you can override the system default in the following ways:

- Use the **-Df** tags of the **onspaces -c -S** command to override the system default for a specific sbspace.
You can later change these sbspace attributes for the sbspace with the **onspaces -ch** option. For more information on valid ranges for the **-Df** tags, refer to the **onspaces** section in the *Administrator's Reference*.
- You override the system default for a specific column when you specify these attributes in the PUT clause of the CREATE TABLE or ALTER TABLE statements.
For more information on these SQL statements, refer to the *IBM Informix Guide to SQL: Syntax*.
- The programmer can override the default values for sbspace attributes for specific smart large objects with functions that the DataBlade API and ESQL/C programming interface provide.

More Information About Sbspaces

Figure 9-13 lists sources of information about various tasks related to using and managing sbspaces.

Figure 9-13
Finding Information for Sbspace Tasks

Task	Reference
Setting memory configuration parameters for smart large objects	Chapter 8, "Managing Shared Memory"
Understanding sbspages	"Sbspages" on page 9-12
Specifying I/O characteristics for an sbspace	onspaces option in "Storage Characteristics of Sbspaces" on page 9-25
Allocating space for an sbspace	"Creating an Sbspace" on page 10-23
Adding a chunk to an sbspace	"Adding a Chunk to an Sbspace" on page 10-26

(1 of 2)

Task	Reference
Defining or altering storage characteristics for a smart large object	“Altering Storage Characteristics of Smart Large Objects” on page 10-27 PUT clause of CREATE TABLE or ALTER TABLE statement in <i>IBM Informix Guide to SQL: Syntax</i>
Monitoring sbspaces	“Monitoring Sbspaces” on page 10-53 Chapter on table performance considerations in <i>Performance Guide</i>
Setting up logging for an sbspace	“Logging Sbspaces and Smart Large Objects” on page 13-13
Backing up an sbspace	“Backing Up Sbspaces” on page 14-7
Checking consistency of an sbspace	“Validating Metadata” on page 21-6
Understanding an sbspace structure	Chapter on disk structures in the <i>Administrator’s Reference</i>
Using onspaces for sbspaces	Chapter on utilities in the <i>Administrator’s Reference</i>
Creating a table with CLOB or BLOB data types	<i>IBM Informix Guide to SQL: Syntax</i>
Accessing smart large objects in an application	<i>IBM Informix DataBlade API Programmer’s Guide</i> <i>IBM Informix ESQ/L/C Programmer’s Manual</i>
Calculating the metadata area size	Chapter on table performance in <i>Performance Guide</i>
Improving metadata I/O	
Changing storage characteristics	
Understanding smart-large-object locking	Chapter on locking in <i>Performance Guide</i>
Configuring sbspaces for temporary smart large objects	Chapter on configuration effects on I/O activity in <i>Performance Guide</i>

(2 of 2)

Temporary Sbspaces

Use a *temporary sbsp* to store temporary smart large objects without metadata logging and user-data logging. If you store temporary smart large objects in a standard sbsp, the metadata is logged. Temporary sbspaces are similar to temporary dbspaces. To create a temporary sbsp, use the **onspaces -c -S** command with the **-t** option. For more information, see [“Creating a Temporary Sbsp” on page 10-27](#).

You can store temporary large objects in a standard sbsp or temporary sbsp.

- If you specify a temporary sbsp in the SBSPACETEMP parameter, you can store temporary smart large objects there.
- If you specify a standard sbsp in the SBSPACENAME parameter, you can store temporary and permanent smart large objects there.
- If you specify a temporary sbsp name in the CREATE TEMP TABLE statement, you can store temporary smart large objects there.
- If you specify a permanent sbsp name in the CREATE TABLE statement, you can store temporary smart large objects there.
- If you omit the SBSPACETEMP and SBSPACENAME parameters and create a smart large object, error message -12053 might display.
- If you specify a temporary sbsp in the SBSPACENAME parameter, you cannot store a *permanent* smart large object in that sbsp. You can store temporary smart large objects in that sbsp.

Comparison of Temporary and Standard Sbspaces

Figure 9-14 compares standard and temporary sbspaces.

Figure 9-14
Temporary and Standard Sbspaces

Characteristics	Standard Sbspace	Temporary Sbspace
Stores smart large objects	Yes	No
Stores temporary smart large objects	Yes	Yes
Logs metadata	Metadata is always logged	Metadata is not logged
Logs user data	User data is not logged for temporary smart large objects but is logged for permanent smart large objects if LOGGING=ON	User data is not logged Creation and deletion of space, and addition of chunks is logged
Fast recovery	Yes	No (the sbspace is emptied when the database server restarts) To initialize shared memory without cleaning up temporary smart large object, specify oninit -p . If you keep temporary large objects, their state is indeterminate.
Backup and restore	Yes	No
Add and drop chunks	Yes	Yes
Configuration parameter	SBSPACENAME	SBSPACETEMP

Temporary Smart Large Objects

Use *temporary smart large objects* to store text or image data (CLOB or BLOB) that do not require restoring from a backup or log replay in fast recovery. Temporary smart large objects last for the user session and are much faster to update than smart large objects.

You create a temporary smart large object in the same way as a permanent smart large object, except you set the LO_CREATE_TEMP flag in the **ifx_lo_specset_flags** or **mi_lo_specset_flags** function. Use **mi_lo_copy** or **ifx_lo_copy** to create a permanent smart large object from a temporary smart large object. For details on creating temporary smart large objects, see the *IBM Informix DataBlade API Programmer's Guide*.

Important: Store pointers to temporary large objects in temporary tables only. If you store them in standard tables and reboot the database server, it results in an error that says that the large object does not exist.

Figure 9-15 compares standard and temporary smart large objects.

Figure 9-15
Temporary and Standard Smart Large Objects

Characteristics	Smart Large Object	Temporary Smart Large Object
Creation flags	LO_CREATE_LOG or LO_CREATE_NOLOG	LO_CREATE_TEMP
Rollback	Yes	No
Logging	Yes, if turned on	No
Duration	Permanent (until user deletes it)	Deleted at end of user session or transaction
Table type stored in	Permanent or temporary table	Temporary tables

Extspaces

An extspace is a logical name associated with an arbitrary string that signifies the location of external data. The resource that the extspace references depends on a user-defined access method for accessing its contents.

For example, a database user might require access to binary files encoded in a proprietary format. First, a developer creates an *access method*, which is a set of routines that access the data. These routines are responsible for all interaction between the database server and the external file. A DBA then adds an extspace that has the file as its target to the database. Once the DBA creates a table in the extspace, the users can access the data in the proprietary files via SQL statements. To locate those files, use the extspace information.

An extspace need not be a filename. For example, it can be a network location. The routines that access the data can use information found in the string associated with the extspace in any manner.

For more information on user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*. For more information on creating functions and primary access methods, see the *IBM Informix Guide to SQL: Syntax*.

Databases

A database is a logical storage unit that contains tables and indexes. (See [“Tables” on page 9-36](#).) Each database also contains a system catalog that tracks information about many of the elements in the database, including tables, indexes, SPL routines, and integrity constraints.

A database resides in the dbspace specified by the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database resides in the root dbspace. When you *do* specify a dbspace in the CREATE DATABASE statement, this dbspace is the location for the following tables:

- Database system catalog tables
- Any table that belongs to the database

Figure 9-16 shows the tables contained in the stores_demo database.

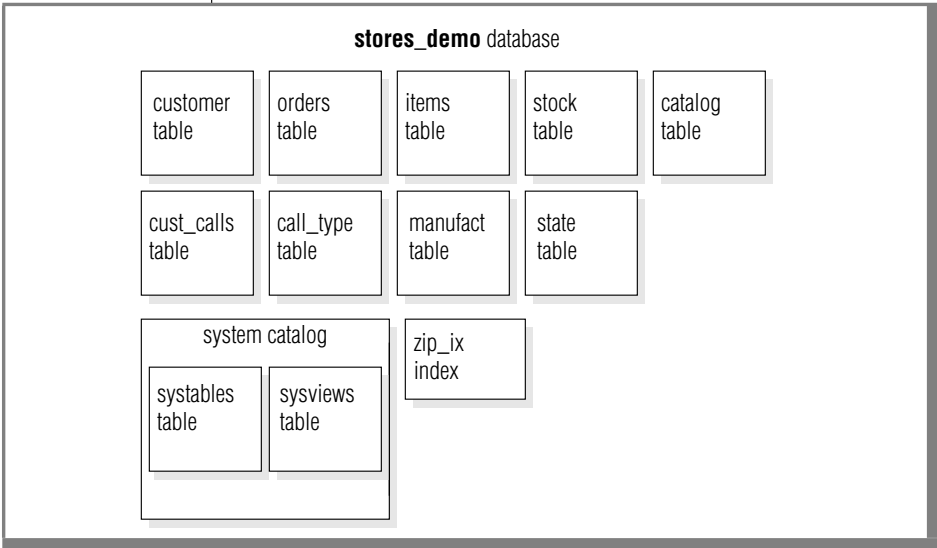


Figure 9-16
The stores_demo Database

The size limits that apply to databases are related to their location in a dbspace. To be certain that all tables in a database are created on a specific physical device, assign only one chunk to the device, and create a dbspace that contains only that chunk. Place your database in that dbspace. When you place a database in a chunk assigned to a specific physical device, the database size is limited to the size of that chunk.

For instructions on how to list the databases that you create, see [“Displaying Databases” on page 10-40](#).

Tables

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

When users create a table, the database server allocates disk space for the table in a block of pages called an extent. (See [“Extents” on page 9-14](#).) You can specify the size of both the first and any subsequent extents.

Users can place the table in a specific dbspace by naming the dbspace when they create the table (usually with the *IN dbspace* option of *CREATE TABLE*). When the user does not specify the dbspace, the database server places the table in the dbspace where the database resides.

Users can also fragment a table over more than one dbspace. Users must define a distribution scheme for the table that specifies which table rows are located in which dbspaces.

A table or table fragment resides completely in the dbspace in which it was created. The database server administrator can use this fact to limit the growth of a table by placing a table in a dbspace and then refusing to add a chunk to the dbspace when it becomes full.

For more information about distribution schemes, see the *IBM Informix Database Design and Implementation Guide*. For information on how to fragment tables and indexes over multiple disks to improve performance, concurrency, data availability, and backups, refer to your *Performance Guide*.

A table, composed of extents, can span multiple chunks, as [Figure 9-17](#) shows.

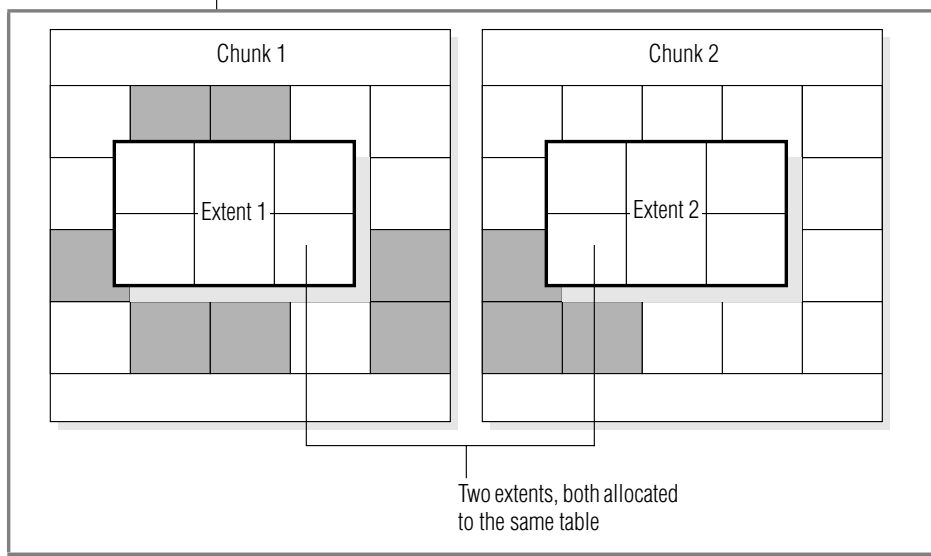


Figure 9-17
Table That Spans
More than One
Chunk

Simple large objects reside in blobpages in either the dbspace with the data pages of the table or in a separate blobspace. When you use the Optical Subsystem, you can also store simple large objects in an optical storage subsystem.

For advice on where to store your tables, see [“Disk-Layout Guidelines” on page 9-52](#) and your *Performance Guide*.

Table Types for Dynamic Server

You can create logging or nonlogging tables in a logging database on Dynamic Server. The two table types are STANDARD (logging tables) and RAW (nonlogging tables). The default standard table is like a table created in earlier versions without a special keyword specified. You can create either a STANDARD or RAW table and change tables from one type to another.

In a nonlogging database, both STANDARD tables and RAW tables are nonlogging. In a nonlogging database, the only difference between STANDARD and RAW tables is that RAW tables do not allow indexes and referential constraints.

[Figure 9-18](#) lists the properties of the types of tables available with Dynamic Server. The flag values are the hexadecimal values for each table type in the **flags** column of **systables**.

Figure 9-18
Table Types for Dynamic Server

Characteristic	STANDARD	RAW	TEMP
Permanent	Yes	Yes	No
Logged	Yes	No	Yes
Indexes	Yes	No	Yes
Rollback	Yes	No	Yes
Recoverable	Yes	Yes, if not updated	No
Restorable	Yes	Yes, if not updated	No

(1 of 2)

Loadable	Yes	Yes	Yes
Enterprise Replication	Yes	No	No
Flag Value	None	0x10	None

(2 of 2)

Standard Permanent Tables

A STANDARD table is the same as a table in a logged database that the database server creates. STANDARD tables do not use light appends. All operations are logged, record by record, so STANDARD tables can be recovered and rolled back. You can back up and restore STANDARD tables. Logging enables updates since the last physical backup to be applied when you perform a warm restore or point-in-time restore. Enterprise Replication is allowed on STANDARD tables.

A STANDARD table is the default type on both logging and nonlogging databases. STANDARD tables are logged if stored in a logging database but are not logged if stored in a nonlogging database.

RAW Tables

RAW tables are nonlogging permanent tables and are similar to tables in a nonlogging database. RAW tables use *light appends*, which add rows quickly to the end of each table fragments. Updates, inserts, and deletes in a RAW table are supported but not logged. RAW tables do not support indexes, referential constraints, or rollback. You can restore a RAW table from the last physical backup if it has not been updated since that backup. Fast recovery rolls back incomplete transactions on STANDARD tables but not on RAW tables. A RAW table has the same attributes whether stored in a logging or nonlogging database.

RAW tables are intended for the initial loading and validation of data. To load RAW tables, you can use any loading utility, including **dbexport** or the High-Performance Loader (HPL) in express mode. If an error or failure occurs while loading a RAW table, the resulting data is whatever was on the disk at the time of the failure.



It is recommended that you do not use RAW tables within a transaction. Once you have loaded the data, use the ALTER TABLE statement to change the table to type STANDARD and perform a level-0 backup before you use the table in a transaction.

Warning: Do not use Enterprise Replication on RAW or TEMP tables.

Temp Tables

Temp tables are temporary, logged tables that are dropped when the user session closes, the database server shuts down, or on reboot after a failure. Temp tables support indexes, constraints, and rollback. You cannot recover, back up, or restore temp tables. Temp tables support bulk operations such as light appends, which add rows quickly to the end of each table fragment. For more information about light appends, refer to your *Performance Guide*.

For more information, see [“Temporary Tables” on page 9-42](#).

Properties of Table Types

This section discusses loading tables, fast recovery, and backup and restore of table types.

Loading of Data Into a Table

Dynamic Server creates STANDARD tables that use logging by default. Data warehousing applications can have huge tables that take a long time to load. Nonlogging tables are faster to load than logging tables. You can use the CREATE RAW TABLE statement to create a RAW table or use the ALTER TABLE statement to change a STANDARD table to RAW before loading the table. After you load the table, run UPDATE STATISTICS on it.

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

Fast Recovery of Table Types

Figure 9-19 shows fast recovery scenarios for the table types available with Dynamic Server.

Figure 9-19
Fast Recovery of Table Types

Table Type	Fast Recovery Behavior
Standard	Fast recovery is successful. All committed log records are rolled forward, and all incomplete transactions are rolled back.
RAW	If a checkpoint completed since the RAW table was modified last, all the data is recoverable. Inserts, updates, and deletions that occurred after the last checkpoint are lost. Incomplete transactions in a RAW table are not rolled back.

Backup and Restore of RAW Tables

Figure 9-20 discusses backup scenarios for the table types available on Dynamic Server.

Figure 9-20
Backing Up Tables on Dynamic Server

Table Type	Backup Allowed?
Standard	Yes.
Temp	No.
RAW	Yes. If you update a RAW table, you must back it up so that you can restore all the data in it. Backing up only the logical logs is not enough.



Important: After you load a RAW table or change a RAW table to type STANDARD, you must perform a level-0 backup.

Figure 9-21 shows restore scenarios for these table types.

Figure 9-21
Restoring Tables on Dynamic Server

Table Type	Restore Allowed?
Standard	Yes. Warm restore, cold restore, and point-in-time restore work.
Temp	No.
RAW	When you restore a RAW table, it contains only data that was on disk at the time of the last backup. Because RAW tables are not logged, any changes that occurred since the last backup are not restored.

Temporary Tables

The database server needs to provide disk space for temporary tables of the following two kinds:

- Temporary tables that you create with an SQL statement, such as `CREATE TEMP TABLE...` or `SELECT INTO SCRATCH`
- Temporary tables that the database server creates as it processes a query

Make sure that your database server has configured enough temporary space for both user-created and database server-created temporary tables. Some uses of the database server might require as much temporary storage space as permanent storage space, or more.

If you want to use the temporary dbspaces specified in the `DBSPACETEMP` configuration parameter or `DBSPACETEMP` environment variable, you must specify the `WITH NO LOG` clause when you use the `SELECT...INTO TEMP` statement.



Important: By default, the database server stores temporary tables in the root dbspace. If you decide not to store your temporary tables in the root dbspace, use the `DBSPACETEMP` environment variable or configuration parameter to specify a list of dbspaces for temporary tables.

Temporary Tables That You Create

You can create temporary tables with any of the following SQL statements:

- TEMP TABLE option of the CREATE TABLE statement
- INTO TEMP clause of the SELECT statement, such as `SELECT * FROM customer INTO TEMP cust_temp`

Only the session that creates a temporary table can use the table. When the session exits, the table is dropped automatically.

When you create a temporary table, the database server uses the following criteria:

- If the query used to populate the TEMP table produces no rows, the database server creates an empty, unfragmented table.
- If the rows that the query produces do not exceed 8 kilobytes, the temporary table resides in only one dbspace.
- If the rows exceed 8 kilobytes, the database server creates multiple fragments and uses a round-robin fragmentation scheme to populate them unless you specify a fragmentation method and location for the table.

Where User-Created Temporary Tables are Stored

If your application lets you specify the location of a temporary table, you can specify either logging spaces or nonlogging spaces that you create exclusively for temporary tables. If you create a TEMP table in a temporary space, you must use the WITH NO LOG keyword. SCRATCH tables are not logged and must be created in temporary spaces.

For information about creating temporary dbspaces and dbslices, refer to the **onspaces** section in the *Administrator's Reference*.

If you do not specify the location of a temporary table, the database server stores the temporary table in one of the spaces that you specify as an argument to the DBSPACETEMP configuration parameter or environment variable. The database server keeps track of the name of the last dbspace that it used for a temporary table. When the database server receives another request for temporary storage space, it uses the next available dbspace to spread I/O evenly across the temporary storage space.

For information about where the database stores temporary tables when you do not list any spaces as an argument to DBSPACETEMP, see the DBSPAC-ETEMP section in the *IBM Informix Dynamic Server Administrator's Reference*.

When you use an application to create a temporary table, you can use the temporary table until the application exits or performs one of the following actions:

- Closes the database in which the table was created and opens a database in a different database server
- Closes the database in which the table was created
- Explicitly drops the temporary table

Temporary Tables That the Database Server Creates

The database server sometimes creates temporary tables while running queries against the database or backing it up. The database server might create a temporary table in any of the following circumstances:

- Statements that include a GROUP BY or ORDER BY clause
- Statements that use aggregate functions with the UNIQUE or DISTINCT keywords
- SELECT statements that use auto-index or hash joins
- Complex CREATE VIEW statements
- DECLARE statements that create a scroll cursor
- Statements that contain correlated subqueries
- Statements that contain subqueries that occur within an IN or ANY clause
- CREATE INDEX statements

When the process that initiated the creation of the table is complete, the database server deletes the temporary tables that it creates.

If the database server shuts down without removing temporary tables, it performs temporary table cleanup the next time shared-memory is initialized. To initialize shared memory without temporary table cleanup, execute **oninit** with the **-p** option.



Important: *In addition to temporary tables, the database server uses temporary disk space to store the before images of data that are overwritten while backups are occurring and overflow from query processing that occurs in memory. Make sure that you have configured enough temporary space for all uses.*

Where Database Server-Created Temporary Tables are Stored

When the database server creates a temporary table, it stores the temporary table in one of the dbspaces that you specify in the DBSPACETEMP configuration parameter or the DBSPACETEMP environment variable. The environment variable supersedes the configuration parameter.

When you do not specify any temporary dbspaces in DBSPACETEMP, or the temporary dbspaces that you specify have insufficient space, the database server creates the table in a standard dbspace according to the following rules:

- If you created the temporary table with CREATE TEMP TABLE, the database server stores this table in the dbspace that contains the database to which the table belongs.
- If you created the temporary table with the INTO TEMP option of the SELECT statement, the database server stores this table in the root dbspace.

For more information, see [“Creating a Temporary Dspace” on page 10-16.](#)

Tbspaces

Database server administrators sometimes need to track disk use by a particular table. A *tblspace* contains all the disk space allocated to a given table or table fragment (if the table is fragmented). A separate *tblspace* contains the disk space allocated for the associated index.

A *tblspace*, for example, does not correspond to any particular part of a chunk or even to any particular chunk. The indexes and data that make up a *tblspace* might be scattered throughout your chunks. The *tblspace*, however, represents a convenient accounting entity for space across chunks devoted to a particular table. (See [“Tables” on page 9-36.](#))

Maximum Number of Tbspaces in a Table

You can specify a maximum of 2^{20} (or 1,048,576) tablespaces in a table.

Table and Index Tbspaces

The table tblspace contains the following types of pages:

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

The index tblspace contains the following types of pages:

- Pages allocated to indexes
- Bitmap pages that track page use within the index extents

Figure 9-22 illustrates the tablespaces for three tables that form part of the **stores_demo** database. Only one table (or table fragment) exists per tblspace. An index resides in a separate tblspace from the associated table. Blobpages represent TEXT or BYTE data stored in a dbspace.

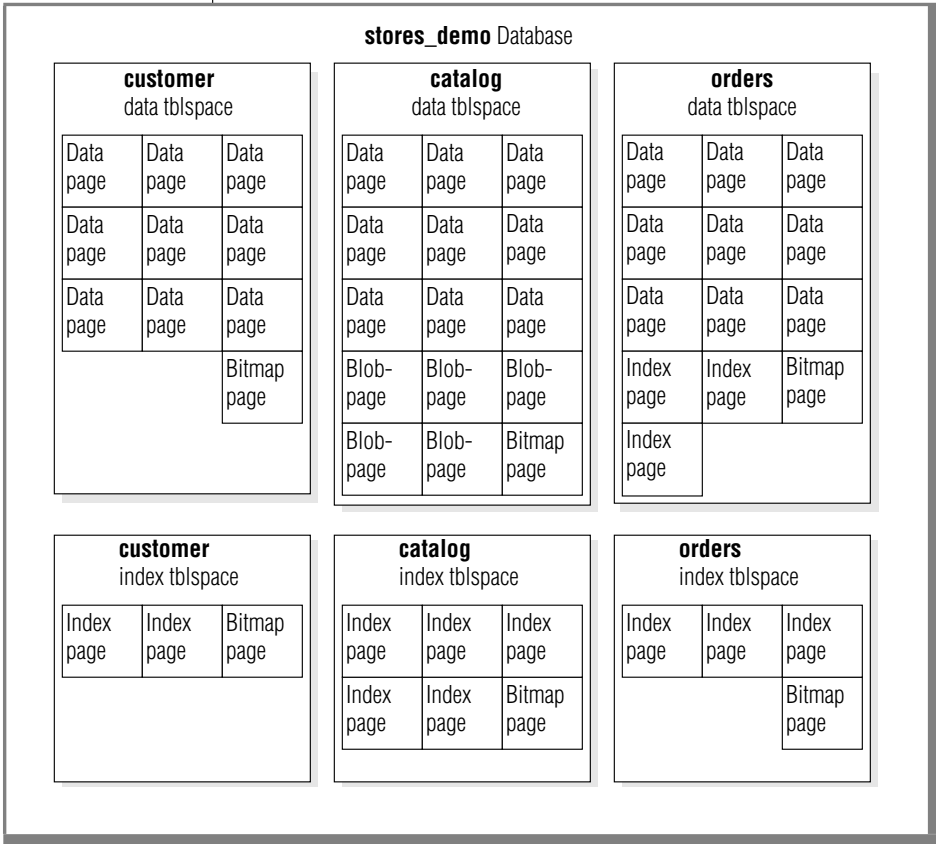


Figure 9-22
Sample Tbspaces in
the *stores_demo*
Database

Extent Interleaving

The database server allocates the pages that belong to a `tblspace` as extents. Although the pages within an extent are contiguous, extents might be scattered throughout the `dbspace` where the table resides (even on different chunks). [Figure 9-23](#) depicts this situation with two noncontiguous extents that belong to the `tblspace` for **table_1** and a third extent that belongs to the `tblspace` for **table_2**. A **table_2** extent is positioned between the first **table_1** extent and the second **table_1** extent. When this situation occurs, the extents are interleaved. Because sequential access searches across **table_1** require the disk head to seek across the **table_2** extent, performance is worse than if the **table_1** extents were contiguous. For instructions on how to avoid and eliminate interleaving extents, see your *Performance Guide*.

Figure 9-23
Three Extents That Belong to Two Different Tblspaces in a Single Dbspace

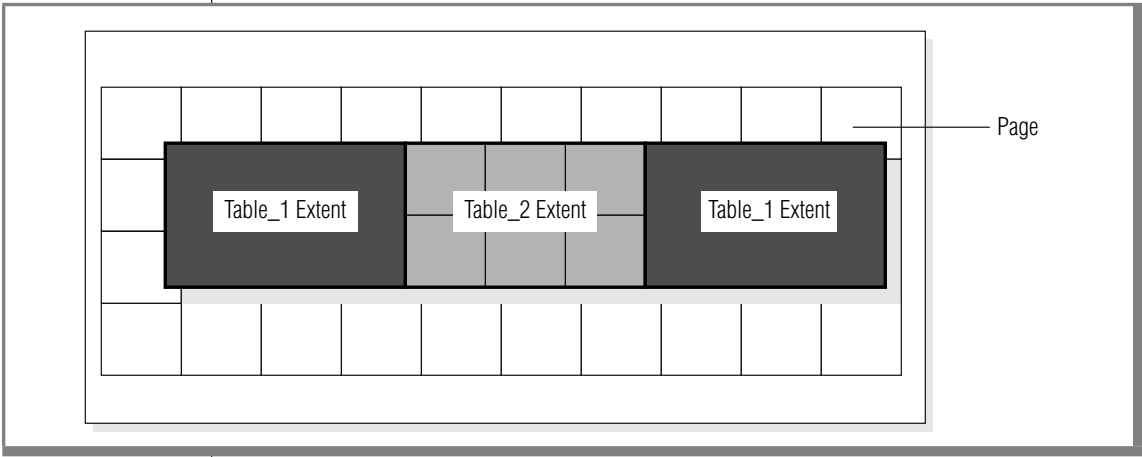


Table Fragmentation and Data Storage

The fragmentation feature gives you additional control over where the database stores data. You are not limited to specifying the locations of individual tables and indexes. You can also specify the location of table and index *fragments*, which are different parts of a table or index that reside on different storage spaces. You can fragment the following storage spaces:

- Dbspaces
- Sbspaces

For more information about fragmentation, see your *Performance Guide*.

Amount of Disk Space Needed to Store Data

To determine how much disk space you need, follow these steps:

To determine how much disk space you need

1. Calculate the size requirements of the root dbspace.
2. Estimate the total amount of disk space to allocate to all the database server databases, including space for overhead and growth.

The following sections explain these steps.

Size of the Root Dbspace

To calculate the size of the root dbspace, take the following storage structures into account:

- The physical log (200 kilobytes minimum)
- The logical-log files (200 kilobytes minimum)
- Temporary tables
- Data
- System databases (**sysmaster**, **sysutils**, **syscdr**, **sysuuid**) and system catalogs (the size varies between versions)

- Reserved pages (~24 kilobytes)
- Tablespace tablespace (100 to 200 kilobytes minimum)
- Extra space

This estimate is the initial root dbspace size before you initialize the database server. The initial size of the root dbspace depends on whether you plan to store the physical log, logical logs, and temporary tables in the root dbspace or in another dbspace. The root dbspace must be large enough for the minimum size configuration during disk initialization.

It is recommended that you initialize the system with a small log size (for example, three 1000-kilobyte log files, or 3000 kilobytes for the total log size). Once initialization is complete, create new dbspaces, move and resize the logical-log files, and drop the original logs in the root dbspace. Then move the physical log to another dbspace. This procedure minimizes the impact of the logs in the root dbspace because:

- A large amount of space is not left unused in the root dbspace after you move the logs (if your root dbspace does not grow).
- The logs do not contend for space and I/O on the same disk as the root dbspace (if your root dbspace does grow).

For details on how to move the logs, see [“Moving a Logical-Log File to Another Dbspace” on page 14-20](#) and [“Changing the Physical-Log Location and Size” on page 16-3](#).

You can add chunks and drop empty chunks in the root dbspace. Start with a small root dbspace and expand it as your system grows. However, you cannot start with a large initial root chunk and shrink it.

Physical and Logical Logs

The value stored in the ONCONFIG parameter PHYSFILE defines the size of your physical log. Advice on sizing your physical log is contained in [“Size and Location of the Physical Log” on page 15-6](#).

To calculate the size of the logical-log files, multiply the value of the ONCONFIG parameter LOGSIZE by the number of logical-log files. For advice on sizing your logical log, see [“Estimating the Size and Number of Log Files” on page 14-4](#).

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that the database server might require for temporary tables. Try to estimate how many of these statements are to run concurrently. The space occupied by the rows and columns that are returned provides a good basis for estimating the amount of space required.

The largest temporary table that the database server creates during a warm restore is equal to the size of your logical log. You calculate the size of your logical log by adding the sizes of all logical-log files.

You must also analyze end-user applications to estimate the amount of disk space that the database server might require for explicit temporary tables.

For more information, including a list of statements that require temporary space, see [“Temporary Tables” on page 9-42](#).

Critical Data

It is recommended that you do not store databases and tables in the root dbspace. Mirror the root dbspace and other dbspaces that contain critical data such as the physical log and logical logs. Estimate the amount of disk space, if any, that you need to allocate for tables stored in the root dbspace.

Extra Space

Allow extra space in the root dbspace for the system databases to grow, for the extended reserved pages, and ample free space. The number of extended reserved pages depends on the number of primary chunks, mirror chunks, logical-log files, and storage spaces in the database server.

Amount of Space That Databases Require

The amount of additional disk space required for the database server data storage depends on the needs of your end users, plus overhead and growth. Every application that your end users run has different storage requirements. The following list suggests some of the steps that you can take to calculate the amount of disk space to allocate (beyond the root dbspace):

- Decide how many databases and tables you need to store. Calculate the amount of space required for each one.
- Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
- Decide which databases and tables you want to mirror.

For instructions about calculating the size of your tables, refer to your *Performance Guide*.

Disk-Layout Guidelines

The following are typical goals for efficient disk layout:

- Limiting disk-head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

You must make some trade-offs among these goals when you design your disk layout. For example, separating the system catalog tables, the logical log, and the physical log can help reduce contention for these resources. However, this action can also increase the chances that you have to perform a system restore. For detailed disk-layout guidelines, see the *Performance Guide*.

Dbospace and Chunk Guidelines

This section lists some general strategies for disk layout that do not require any information about the characteristics of a particular database:

- Associate disk partitions with chunks and allocate at least one *additional* chunk for the root dbospace.

A disk that is already partitioned might require the use of offsets. For details, see [“Allocating Raw Disk Space on UNIX” on page 10-9](#).



Tip: *With the 4-terabyte maximum size of a chunk, you can avoid partitioning by assigning a chunk per disk drive.*

- Mirror critical dbospaces: the root dbospace, the dbospaces that contain the physical log and the logical-log files. Also mirror high-use databases and tables.

You specify mirroring at the dbospace level. Mirroring is either on or off for all chunks belonging to a dbospace. Locate the primary and the mirrored dbospaces on different disks. Ideally, different controllers handle the different disks.

- Spread temporary tables and sort files across multiple disks.

To define several dbospaces for temporary tables and sort files, use **onspaces -t**. When you place these dbospaces on different disks and list them in the DBSPACETEMP configuration parameter, you can spread the I/O associated with temporary tables and sort files across multiple disks. For information on using the DBSPACETEMP configuration parameter or environment variable, see the chapter on configuration parameters in the *Administrator's Reference*.

- Keep the physical log in the root dbospace but move the logical logs from the root dbospace. However, if you plan to store the system catalogs in the root dbospace, move the physical log to another dbospace.

For advice on where to store your logs, see [“Specifying the Location of the Physical Log” on page 15-6](#) and [“Location of Logical-Log Files” on page 13-4](#). Also see [“Moving a Logical-Log File to Another Dbospace” on page 14-20](#) and [“Changing the Physical-Log Location and Size” on page 16-3](#).

- To improve backup and restore performance:
 - Cluster system catalogs with the data that they track.
 - If you use ON-Bar to perform parallel backups to a high-speed tape drive, store the databases in several small dbspaces.
- For additional performance recommendations, see the *IBM Informix Backup and Restore Guide*.

Table-Location Guidelines

This section lists some strategies for optimizing the disk layout, given certain characteristics about the tables in a database. You can implement many of these strategies with a higher degree of control using table fragmentation:

- Isolate high-use tables on a separate disk.

To isolate a high-use table on its own disk device, assign the device to a chunk, and assign the same chunk to a dbspace. Finally, place the frequently used table in the dbspace just created using the *IN dbspace* option of **CREATE TABLE**.

To display the level of I/O operations against each chunk, run the **onstat -g iof** option.
- Fragment high-use tables over multiple disks.
- Group related tables in a dbspace.

If a device that contains a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. Although you must perform a cold restore if a dbspace that contains critical information fails, you need only perform a warm restore if a noncritical dbspace fails.
- Place high-use tables on the middle partition of a disk.
- Optimize table extent sizes.

For more information, see the chapter on table performance considerations in your *Performance Guide*. For information on **onstat** options, see the *Administrator's Reference*.

Sample Disk Layouts

When setting out to organize disk space, the database server administrator usually has one or more of the following objectives in mind:

- High performance
- High availability
- Ease and frequency of backup and restore

Meeting any one of these objectives has trade-offs. For example, configuring your system for high performance usually results in taking risks regarding the availability of data. The sections that follow present an example in which the database server administrator must make disk-layout choices given limited disk resources. These sections describe two different disk-layout solutions. The first solution represents a performance optimization, and the second solution represents an availability-and-restore optimization.

The setting for the sample disk layouts is a fictitious sporting goods database that uses the structure (but not the volume) of the **stores_demo** database. In this example, the database server is configured to handle approximately 350 users and 3 gigabytes of data. The disk space resources are shown in the following table.

Disk Drive	Size of Drive	High Performance
Disk 1	2.5 gigabytes	No
Disk 2	3 gigabytes	Yes
Disk 3	2 gigabytes	Yes
Disk 4	1.5 gigabytes	No

The database includes two large tables: **cust_calls** and **items**. Assume that both of these tables contain more than 1,000,000 rows. The **cust_calls** table represents a record of all customer calls made to the distributor. The **items** table contains a line item of every order that the distributor ever shipped.

The database includes two high-use tables: **items** and **orders**. Both of these tables are subject to constant access from users around the country.

The remaining tables are low-volume tables that the database server uses to look up data such as postal code or manufacturer.

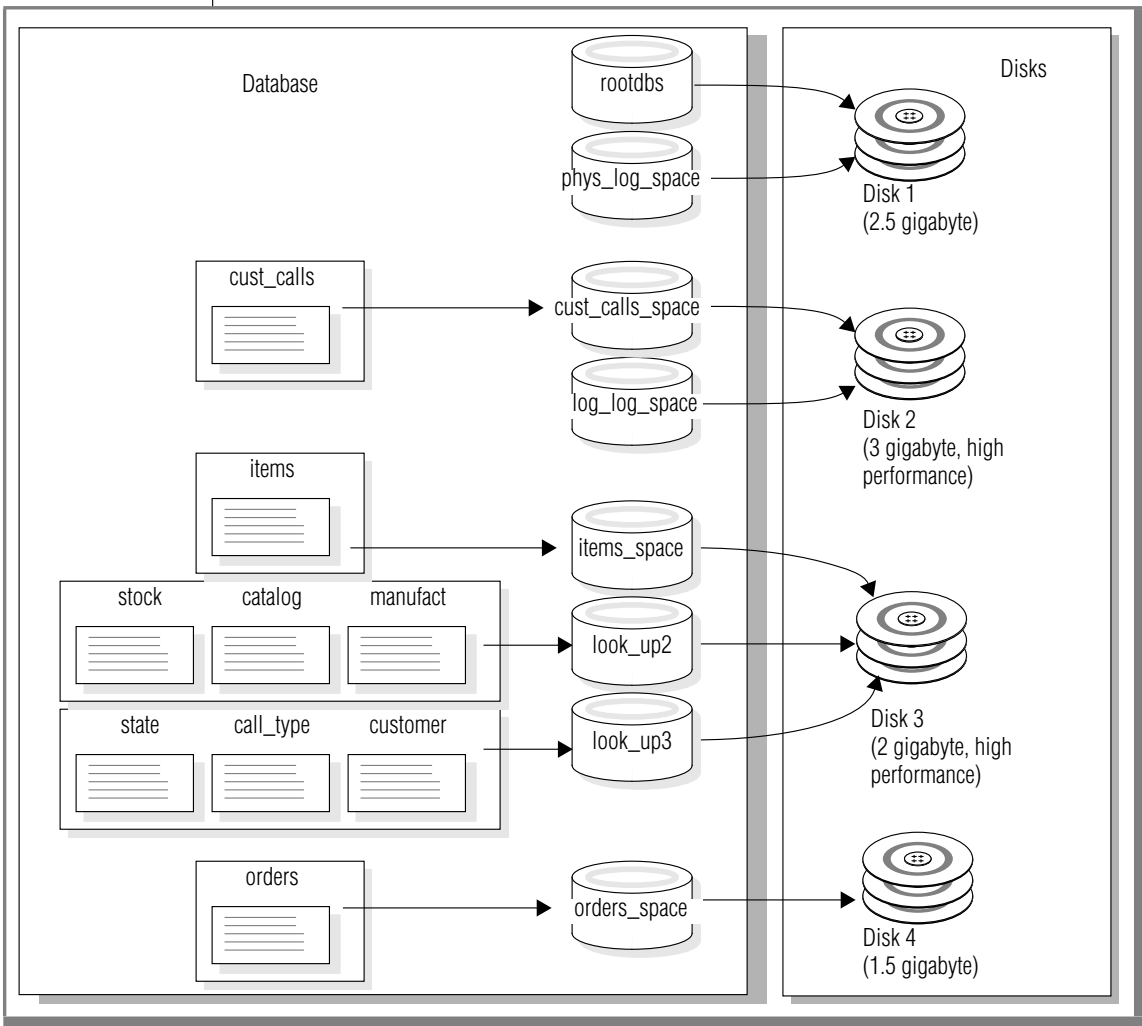
Table Name	Maximum Size	Access Rate
cust_calls	2.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample Layout When Performance Is Highest Priority

[Figure 9-24 on page 9-57](#) shows a disk layout optimized for performance. This disk layout uses the following strategies to improve performance:

- Migration of the logical log from the rootdbs dbspace to a dbspace on a separate disk
This strategy separates the logical log and the physical log and reduces contention for the root dbspace.
- Location of the two tables that undergo the highest use in dbspaces on separate disks
Neither of these disks stores the logical log or the physical log. Ideally you could store each of the **items** and **orders** tables on a separate high-performance disk. However, in the present scenario, this strategy is not possible because one of the high-performance disks is needed to store the very large **cust_calls** table (the other two disks are too small for this task).

Figure 9-24
Disk Layout Optimized for Performance

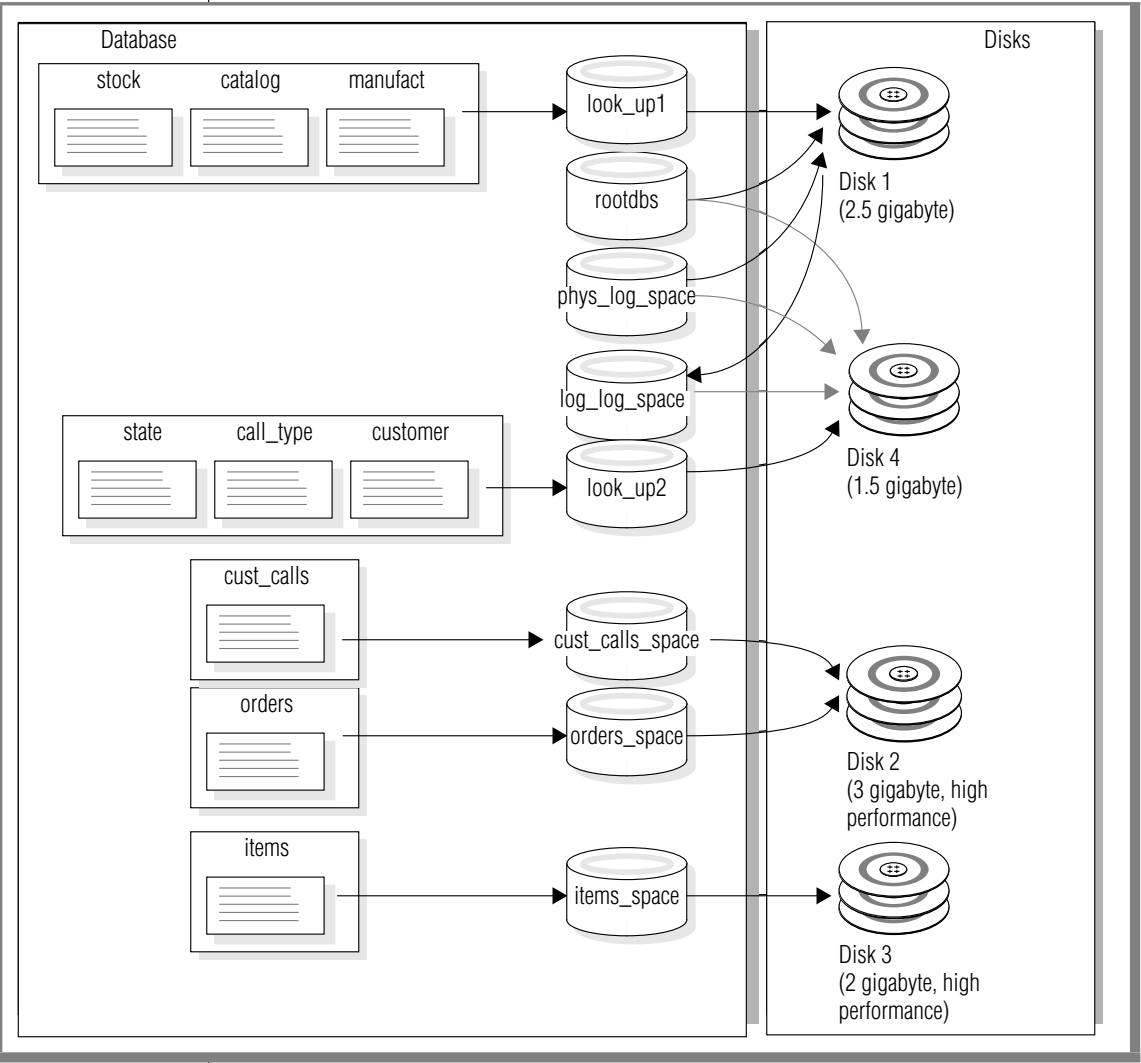


Sample Layout When Availability Is Highest Priority

The weakness of the previous disk layout is that if either Disk 1 or Disk 2 fails, the whole database server goes down until you restore the dbspaces on these disks from backups. In other words, the disk layout is poor with respect to availability.

An alternative disk layout that optimizes for availability is shown in [Figure 9-25 on page 9-59](#). This layout mirrors all the critical data spaces (the system catalog tables, the physical log, and the logical log) to a separate disk. Ideally you could separate the logical log and physical log (as in the previous layout) and mirror each disk to its own mirror disk. However, in this scenario, the required number of disks does not exist; therefore, the logical log and the physical log both reside in the root dbspace.

Figure 9-25
Disk Layout Optimized for Availability



Logical-Volume Manager

A logical-volume manager (LVM) is a utility that allows you to manage your disk space through user-defined logical volumes.

Many computer manufacturers ship their computers with a proprietary LVM. You can use the database server to store and retrieve data on disks that are managed by most proprietary LVMs. Logical-volume managers provide some advantages and some disadvantages, as discussed in the remainder of this section.

Most LVMs can manage multiple gigabytes of disk space. The database server chunks are limited to a size of 4 terabytes, and this size can be attained only when the chunk being allocated has an offset of zero. Consequently, you should limit the size of any volumes to be allocated as chunks to a size of 4 terabytes.

Because LVMs allow you to partition a disk drive into multiple volumes, you can control where data is placed on a given disk. You can improve performance by defining a volume that consists of the middle-most cylinders of a disk drive and placing high-use tables in that volume. (Technically, you do not place a table directly in a volume. You must first allocate a chunk as a volume, then assign the chunk to a dbspace, and finally place the table in the dbspace. For more information, see [“Control of Where Data Is Stored” on page 9-16.](#))



Tip: *If you choose to use large disk drives, you can assign a chunk to one drive and eliminate the need to partition the disk.*

You can also improve performance by using a logical volume manager to define a volume that spreads across multiple disks and then placing a table in that volume.

Many logical volume managers also allow a degree of flexibility that standard operating-system format utilities do not. One such feature is the ability to reposition logical volumes after you define them. Thus getting the layout of your disk space right the first time is not so critical as with operating-system format utilities.

LVMs often provide operating-system-level mirroring facilities. For more information, see [“Alternatives to Mirroring” on page 17-6.](#)

Figure 9-26 illustrates the role of fragments in specifying the location of data.

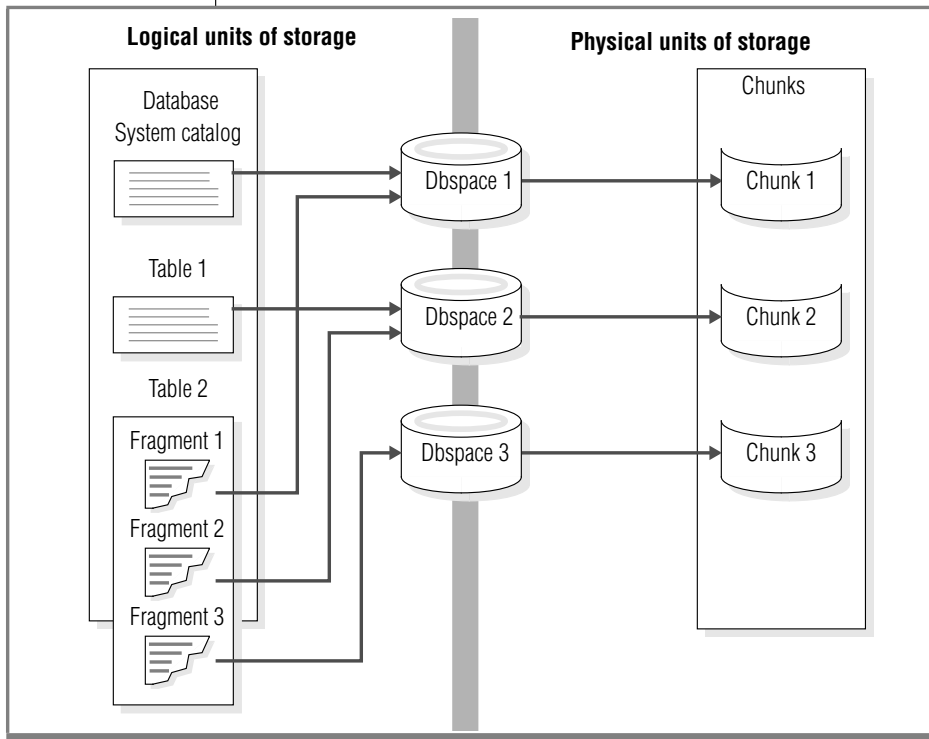


Figure 9-26
Dbspaces That Link
Logical Units
(Including Table
Fragments) and
Physical Units of
Storage

Usually you fragment a table when you initially create it. The CREATE TABLE statement takes one of the following forms:

```
CREATE TABLE tablename ... FRAGMENT BY ROUND ROBIN IN dbspace1,
dbspace2, dbspace3;
```

```
CREATE TABLE tablename ...FRAGMENT BY EXPRESSION
<Expression 1> in dbspace1,
<Expression 2> in dbspace2,
<Expression 3> in dbspace3;
```

The FRAGMENT BY ROUND ROBIN and FRAGMENT BY EXPRESSION keywords refer to two different distribution schemes. Both statements associate fragments with dbspaces. For more information on fragmentation schemes, refer to the *IBM Informix Database Design and Implementation Guide*.

Managing Disk Space

In This Chapter	10-5
Allocating Disk Space	10-6
Specifying an Offset	10-6
Specifying an Offset for the Initial Chunk of Root Dbspace	10-7
Specifying an Offset for Additional Chunks	10-7
Using Offsets to Create Multiple Chunks	10-7
Allocating Cooked File Spaces on UNIX	10-8
Allocating Raw Disk Space on UNIX	10-9
Creating Symbolic Links to Raw Devices	10-10
Allocating NTFS File Space on Windows	10-10
Allocating Raw Disk Space on Windows	10-11
Specifying Names for Storage Spaces and Chunks	10-12
Specifying the Maximum Size of Chunks	10-13
Specifying the Maximum Number of Chunks and Storage Spaces	10-13
Backing Up After You Change the Physical Schema	10-13
Managing Dbspaces	10-14
Creating a Dbspace	10-14
Creating a Temporary Dbspace	10-16
What to Do If You Run Out of Disk Space.	10-17
Adding a Chunk to a Dbspace or Blobspace	10-17
Adding a Chunk with ON-Monitor	10-18
Managing Blobspaces	10-19
Creating a Blobspace	10-19
Preparing Blobspaces to Store TEXT and BYTE Data	10-21
Determining Blobpage Size.	10-21
Determining Database Server Page Size	10-22
Obtaining Blobspace Storage Statistics	10-22

Managing Sbspaces	10-23
Creating an Sbspace	10-23
Sizing Sbspace Metadata.	10-25
Adding a Chunk to an Sbspace	10-26
Altering Storage Characteristics of Smart Large Objects	10-27
Creating a Temporary Sbspace	10-27
Dropping a Chunk	10-29
Verifying Whether a Chunk Is Empty	10-29
Dropping a Chunk from a Dbspace with onspaces.	10-30
Dropping a Chunk from a Blobspace	10-30
Dropping a Chunk from an Sbspace with onspaces	10-30
Using the -f (Force) Option	10-31
Deleting Smart Large Objects Without any Pointers	10-31
Dropping a Storage Space	10-31
Preparing to Drop a Storage Space	10-32
Dropping a Mirrored Storage Space.	10-32
Dropping a Storage Space with onspaces	10-32
Dropping a Dbspace or Blobspace with ON-Monitor	10-33
Backing Up After Dropping a Storage Space	10-33
Managing Extspaces	10-34
Creating an Extspace	10-34
Dropping an Extspace	10-35
Skipping Inaccessible Fragments	10-35
Using the DATASKIP Configuration Parameter.	10-35
Using the Dataskip Feature of onspaces	10-36
Using onstat to Check Dataskip Status	10-36
Using the SQL Statement SET DATASKIP.	10-36
Effect of the Dataskip Feature on Transactions	10-37
Determining When to Use Dataskip.	10-38
Determining When to Skip Selected Fragments	10-38
Determining When to Skip All Fragments	10-39
Monitoring Fragmentation Use	10-39

Displaying Databases	10-40
Using SMI Tables	10-40
Using ISA	10-40
Using ON-Monitor	10-40
Monitoring Disk Usage	10-41
Monitoring Chunks	10-41
onstat -d	10-41
onstat -d update	10-43
onstat -D	10-43
onstat -g iof	10-44
oncheck -pr	10-44
oncheck -pe	10-45
Using IBM Informix Server Administrator	10-46
Using ON-Monitor	10-47
Using SMI Tables	10-48
Monitoring Tblspaces and Extents	10-48
Using SMI Tables	10-49
Monitoring Simple Large Objects in a Blobspace	10-49
onstat -O	10-49
Determining Blobpage Fullness with oncheck -pB	10-51
Monitoring Blobspace Usage with oncheck -pe	10-52
Monitoring Simple Large Objects in a Dbspace with oncheck -pT	10-52
Monitoring Sbspaces	10-53
Using onstat -d	10-55
Using oncheck -ce and oncheck -pe	10-56
Using oncheck -cs	10-57
Using oncheck -ps	10-58
Monitoring the Metadata and User-Data Areas	10-59
Using onstat -g smb c	10-60
Loading Data Into a Table	10-61

In This Chapter

This chapter provides the instructions on managing effectively the disk spaces and data that the database server controls. It assumes you are familiar with the terms and concepts contained in [Chapter 9, “Data Storage.”](#)

You can use the following utilities to manage storage spaces:

- **onspaces**
- ISA

This chapter covers the following topics:

- Allocating disk space
- Specifying chunk names, and the maximum size and number of chunks and storage spaces
- Backing up after you change the physical schema
- Creating and dropping dbspaces, blobspaces, and sbspaces
- Adding, and dropping chunks from dbspaces, blobspaces, and sbspaces
- Creating and dropping extspaces
- Skipping inaccessible fragments
- Monitoring disk space usage
- Monitoring simple-large-object data and smart-large-object data

Your *Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents, how to reclaim space in an empty extent, and how to improve disk I/O.

UNIX

Windows

Allocating Disk Space

This section explains how to allocate disk space for the database server. Read the following sections before you allocate disk space:

- [“Unbuffered or Buffered Disk Access on UNIX” on page 9-8](#)
- [“Amount of Disk Space Needed to Store Data” on page 9-49](#)
- [“Disk-Layout Guidelines” on page 9-52](#)

Before you can create a storage space or chunk, or mirror an existing storage space, you must allocate disk space for the chunk file. You can allocate either an empty file or a portion of raw disk for database server disk space.

On UNIX, if you allocate raw disk space, it is recommended that you use the UNIX **ln** command to create a link between the character-special device name and another filename. For more information on this topic, see [“Creating Symbolic Links to Raw Devices” on page 10-10](#).

Using a UNIX file and its inherent operating-system interface for database server disk space also is referred to as using *cooked space*. ♦

On Windows, it is recommended that you use NTFS files for database server disk space. For more information on this recommendation, see [“Unbuffered or Buffered Disk Access on UNIX” on page 9-8](#). ♦

Specifying an Offset

When you allocate a chunk of disk space to the database server, specify an offset for one of the following two purposes:

- To prevent the database server from overwriting the partition information
- To define multiple chunks on a partition, disk device, or cooked file

The maximum value for the offset is 4 terabytes.

Many computer systems and some disk-drive manufacturers keep information for a physical disk drive on the drive itself. This information is sometimes referred to as a *volume table of contents* (VTOC) or disk label. The VTOC is commonly stored on the first track of the drive. A table of alternate sectors and bad-sector mappings (also called a *revectoring table*) might also be stored on the first track.

If you plan to allocate partitions at the start of a disk, you might need to use offsets to prevent the database server from overwriting critical information required by the operating system. For the exact offset required, refer to your disk-drive manuals.



Warning: *If you are running two or more instances of the database server, be extremely careful not to define chunks that overlap. Overlapping chunks can cause the database server to overwrite data in one chunk with unrelated data from an overlapping chunk. This overwrite effectively destroys overlapping data.*

Specifying an Offset for the Initial Chunk of Root Dbspace

For the initial chunk of root dbspace and its mirror, if it has one, specify the offsets with the ROOTOFFSET and MIRROROFFSET parameters, respectively. For more information, see the chapter on configuration parameters in the *Administrator's Reference*.

Specifying an Offset for Additional Chunks

To specify an offset for additional chunks of database server space, you must supply the offset as a parameter when you assign the space to the database server. For more information, see [“Creating a Dbspace” on page 10-14](#).

Using Offsets to Create Multiple Chunks

You can create multiple chunks from a disk partition, disk device, or file, by specifying offsets and assigning chunks that are smaller than the total space available. The offset specifies the beginning location of a chunk. The database server determines the location of the last byte of the chunk by adding the size of the chunk to the offset.

UNIX

For the first chunk, assign any initial offset, if necessary, and specify the size as an amount that is less than the total size of the allocated disk space. For each additional chunk, specify the offset to include the sizes of all previously assigned chunks, plus the initial offset, and assign a size that is less than or equal to the amount of space remaining in the allocation.

Allocating Cooked File Spaces on UNIX

The following procedure shows an example of allocating a cooked file `usr/data/my_chunk` for disk space on UNIX.

To allocate cooked file space

1. Log in as user **informix**:

```
su informix
```

2. Change directories to the directory where the cooked space will reside:

```
cd /usr/data
```

3. Create your chunk by concatenating null to the filename that the database server will use for disk space:

```
cat /dev/null > my_chunk
```

4. Set the file permissions to 660 (rw-rw----):

```
chmod 660 my_chunk
```

5. You must set both group and owner of the file to **informix**:

```
ls -l my_chunk -rw-rw----
1 informix informix
0 Oct 12 13:43 my_chunk
```

6. Use **onspaces** to create the storage space or chunk.

For information on how to create a storage space using the file you have allocated, refer to [“Creating a Dbspace” on page 10-14](#), [“Creating a Blobspace” on page 10-19](#), and [“Creating an Sbspace” on page 10-23](#).

UNIX

Allocating Raw Disk Space on UNIX

To allocate raw space, you must have a disk partition available that is dedicated to raw space. To create raw disk space, you can either repartition your disks or unmount an existing file system. Back up any files before you unmount the device.

To allocate raw disk space

1. Create and install a raw device.

For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation and [“Unbuffered or Buffered Disk Access on UNIX” on page 9-8](#).

2. Change the ownership and permissions of the character-special devices to **informix**.

The filename of the character-special device usually begins with the letter *r*. For the procedure, see steps 4 and 5 in [“Allocating Cooked File Spaces on UNIX” on page 10-8](#).

3. Verify that the operating-system permissions on the character-special devices are `crw-rw----`.
4. Create a symbolic link between the character-special device name and another filename with the UNIX link command, **ln -s**. For details, see [“Creating Symbolic Links to Raw Devices” on page 10-10](#).



Warning: After you create the raw device that the database server uses for disk space, do not create file systems on the same raw device that you allocate for the database server disk space. Also, do not use the same raw device as swap space that you allocate for the database server disk space.

UNIX

Creating Symbolic Links to Raw Devices

Use symbolic links to assign standard device names and to point to the device. To create a link between the character-special device name and another filename, use the UNIX link command (usually **ln**). To verify that both the devices and the links exist, execute the UNIX command **ls -l** (**ls -lg** on BSD) on your device directory. The following example shows links to raw devices. If your operating system does not support symbolic links, hard links also work.

```
ln -s /dev/rxy0h /dev/my_root # orig_device link to symbolic_name
ln -s /dev/rxy0a /dev/raw_dev2
ls -l
crw-rw--- /dev/rxy0h
crw-rw--- /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Why use symbolic links? If you create chunks on a raw device and that device fails, you cannot restore from a backup until you replace the raw device and use the same pathname. All chunks that were accessible at the time of the last backup must be accessible when you perform the restore.

Symbolic links simplify recovery from disk failure and enable you to replace quickly the disk where the chunk is located. You can replace a failed device with another device, link the new device pathname to the same filename that you previously created for the failed device, and restore the data. You do not need to wait for the original device to be repaired.

Windows

Allocating NTFS File Space on Windows

On Windows, the database server uses NTFS files by default. The NTFS file system allows you to use standard filenames for unbuffered files.

To allocate NTFS file space for database server disk space or mirrored space, the first step is to create a null (zero bytes) file.

To allocate NTFS file space for a dbspace, blobspace, or sbospace

1. Log in as a member of the **Informix-Admin** group.
2. Open an MS-DOS command shell.
3. Change to the directory where you want to allocate the space, as in the following example:

```
c:> cd \usr\data
```

4. Create a null file with the following command:

```
c:> copy nul my_chunk
```

5. If you want to verify that the file has been created, use the **dir** command to do so.

Once you have allocated the file space, you can create the dbspace or other storage space as you normally would, using **onspaces**. For information on how to create a dbspace or a blobspace, refer to [“Creating a Dbspace” on page 10-14](#) and [“Creating a Blobspace” on page 10-19](#).

Windows**Allocating Raw Disk Space on Windows**

You can configure raw disk space on Windows as a logical drive or physical drive. To find the drive letter or disk number, run the **Disk Administrator**. If the drives need to be striped (multiple physical disks combined into one logical disk), only logical drive specification would work.

You must be a member of the **Informix-Admin** group when you create a storage space or add a chunk. The raw disk space can be formatted or unformatted disk space.

To specify a logical drive

1. Assign a drive letter to the disk partition.
2. Specify the following value for ROOTDBS in the ONCONFIG file:

```
\\.\drive_letter
```

3. To create a storage space or add a chunk, specify the logical drive partition.

This example adds a chunk of 5000 kilobytes on the **e:** drive, at an offset of 5200 kilobytes, to dbspace **dpspc3**.

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
```



To specify a physical drive

1. If the disk partition has *not* been assigned a drive letter, specify the following value for ROOTDBS in the ONCONFIG file:

```
\\.\PhysicalDrive<number>
```

2. To create a storage space or add a chunk, specify the physical drive partition.

This example adds a chunk of 5000 kilobytes on **PhysicalDrive0**, at an offset of 5200 kilobytes, to dbspace **dbspc3**.

```
onspaces -a dbspc3 \\.\PhysicalDrive0 : -o 5200 -s 5000
```

Warning: If you allocate a formatted drive or disk partition as raw disk space and it contains data, the database server overwrites the data when it begins to use the disk space. You must ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.

Specifying Names for Storage Spaces and Chunks

Chunk names follow the same rules as storage-space names. Specify an explicit pathname for a storage space or chunk as follows:

- If you are using raw disks on UNIX, it is recommended that you use a linked pathname. (See [“Creating Symbolic Links to Raw Devices” on page 10-10.](#)) ♦
- If you are using raw disks on Windows, the pathname takes the following form, where *x* specifies the disk drive or partition:

```
\\.\x:
```

♦
- If you are using a file for database server disk space, the pathname is the complete path and filename.

Use these naming rules when you create storage spaces or add a chunk. The filename must have the following characteristics:

- Be unique and not exceed 128 characters
- Begin with a letter or underscore
- Contain only letters, digits, underscores, or \$ characters

UNIX

Windows

The name is case insensitive unless you use quotes around it. By default, the database server converts uppercase characters in the name to lowercase. If you want to use uppercase in names, put quotes around them and set the `DELIMIDENT` environment variable to `ON`.

Specifying the Maximum Size of Chunks

On most platforms, the maximum chunk size is 4 terabytes, but on other platforms, the maximum chunk size is 8 terabytes. To determine which chunk size your platform supports refer to your machine notes file. If you do not run `onmode -BC`, then the maximum chunk size is 2 GB.

Specifying the Maximum Number of Chunks and Storage Spaces

You can specify a maximum of 32,766 chunks for a storage space, and a maximum of 32,766 storage spaces on the database server system. The storage spaces can be any combination of `dbspaces`, `blobspaces`, and `sbspaces`. The maximum size of an instance of the server is approximately 128 petabytes. You must run `onmode -BC` to enable the maximum for all of the above.

Backing Up After You Change the Physical Schema

You must perform a level-0 backup of the root `dbspace` and the modified storage spaces to ensure that you can restore the data when you:

- Add or drop mirroring
- Drop a logical-log file
- Change the size or location of the physical log
- Change your storage-manager configuration
- Add, move, or drop a `dbspace`, `blobspace`, or `sbspace`
- Add, move, or drop a chunk to a `dbspace`, `blobspace`, or `sbspace`



Important: When you add a new logical log, you no longer need to perform a level-0 backup of the root dbspaces and modified dbspaces to use the new logical log. However, it is recommended that you perform the level-0 backup to prevent level-1 and level-2 backups from failing.

You must perform a level-0 backup of the modified storage spaces to ensure that you can restore the unlogged data before you switch to a logging table type:

- When you convert a nonlogging database to a logging database
- When you convert a RAW table to standard

Managing Dbspaces

This section discusses creating a standard or temporary dspace and adding a chunk to a dspace or blobpace.

Creating a Dspace

This section explains how to use **onspaces** to create a standard dspace and a temporary dspace. For information on using ISA to create a dspace, see the ISA online help.

The newly added dspace (and its mirror, if one exists) is available immediately. If you are using mirroring, you can mirror the dspace when you create it. Mirroring takes effect immediately.

To create a standard dspace using onspaces

1. On UNIX, you must be logged in as user **informix** or **root** to create a dspace.
On Windows, users in the **Informix-Admin** group can create a dspace.
2. Ensure that the database server is in online or quiescent mode.
3. Allocate disk space for the dspace, as described in [“Allocating Disk Space” on page 10-6](#).

4. To create a dbspace, use the **onspaces -c -d** options.
Kilobytes is the default unit for the **-s size** and **-o offset** options. To convert kilobytes to megabytes, multiply the unit by 1024 (for example, 10 MB = 10 * 1024 KB).
5. After you create the dbspace, you must perform a level-0 backup of the root dbspace and the new dbspace.

The following example creates a 10-megabyte mirrored dbspace, **dbspce1**, with an offset of 5000 kilobytes for both the primary and mirrored chunks, using raw disk space on UNIX:

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10240 -m /dev/raw_dev2
5000
```

The following example creates a 5-megabyte dbspace, **dbspc3**, with an offset of 200 kilobytes, from raw disk space (drive e :) on Windows:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5120
```

For reference information on creating a dbspace with onspaces, see the utilities chapter in the *Administrator's Reference* and [“Dbspaces” on page 9-16](#).

UNIX

To create a dbspace with ON-Monitor

1. Select the **Dbspaces→Create** option.
2. Enter the name of the new dbspace in the field **Dbspace Name**.
3. If you want to create a mirror for the initial dbspace chunk, enter Y in the **Mirror** field.
Otherwise, enter N.
4. If the dbspace that you are creating is a temporary dbspace, enter Y in the **Temp** field.
Otherwise, enter N.
5. Enter the full pathname for the initial primary chunk of the dbspace in the **Full Pathname** field of the primary-chunk section.
6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in kilobytes, in the **Size** field.
8. If you are mirroring this dbspace, enter the mirrored-chunk full pathname, size, and optional offset in the mirrored-chunk section of the screen.

For more information, refer to the ON-Monitor chapter in the *Administrator's Reference*.

Creating a Temporary Dbspace

To specify where to allocate the temporary files, create temporary dbspaces.

To define temporary dbspaces

1. Use the `onspaces` utility with the `-c -d -t` options.
For more information, refer to [“Creating a Dbspace” on page 10-14](#).
2. Use the `DBSPACETEMP` environment variables or the `DBSPACETEMP` configuration parameter to specify the dbspaces that the database server can use for temporary storage.
For further information on `DBSPACETEMP`, refer to the chapter on configuration parameters in the *Administrator's Reference*.
3. If you create more than one temporary dbspace, the dbspaces should reside on separate disks to optimize the I/O.

If you are creating a temporary dbspace, you must make the database server aware of the existence of the newly created temporary dbspace by setting the `DBSPACETEMP` configuration variable, the `DBSPACETEMP` environment variable, or both. The database server does not begin to use the temporary dbspace until you take both of the following steps:

- Set the `DBSPACETEMP` configuration parameter, the `DBSPACETEMP` environment variable, or both.
- Reinitialize the database server.

The following example creates a 5-megabyte temporary dbspace named **temp_space** with an offset of 5000 kilobytes:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5120
```

For more information, see [“Temporary Dbspaces” on page 9-19](#).

What to Do If You Run Out of Disk Space

When the initial chunk of the dbspace that you are creating is a cooked file on UNIX or an NTFS file on Windows, the database server verifies that the disk space is sufficient for the initial chunk. If the size of the chunk is greater than the available space on the disk, a message is displayed and no dbspace is created. However, the cooked file that the database server created for the initial chunk is not removed. Its size represents the space left on your file system before you created the dbspace. Remove this file to reclaim the space.

Adding a Chunk to a Dbspace or Blobspace

You add a *chunk* when a dbspace, blobspace, or sbspace is becoming full or it needs more disk space. Use **onspaces** or ISA to add a chunk. For information on using ISA to add a chunk, see the ISA online help.



Important: *The newly added chunk (and its associated mirror, if one exists) is available immediately. If you are adding a chunk to a mirrored storage space, you must also add a mirrored chunk.*

To add a chunk using onspaces

1. On UNIX, you must be logged in as user **informix** or **root** to add a chunk.
On Windows, users in the **Informix-Admin** group can add a chunk.
2. Ensure that the database server is in online, quiescent mode, or the cleanup phase of fast-recovery mode.
3. Allocate disk space for the chunk, as described in [“Allocating Disk Space” on page 10-6](#).
4. To add a chunk, use the **-a** option of **onspaces**.
If the storage space is mirrored, you must specify the pathname of both a primary chunk and mirrored chunk.
If you specify an incorrect pathname, offset, or size, the database server does not create the chunk and displays an error message. Also see [“What to Do If You Run Out of Disk Space” on page 10-17](#).
5. After you create the chunk, you must perform a level-0 backup of the root dbspace and the dbspace, blobspace, or sbspace that contains the chunk.

The following example adds a 10-megabyte mirrored chunk to **blobsp3**. An offset of 200 kilobytes for both the primary and mirrored chunk is specified. If you are not adding a mirrored chunk, you can omit the **-m** option.

```
onspaces -a blobsp3 -p /dev/raw_dev1 -o 200 -s 10240 -m /dev/raw_dev2 200
```

The next example adds a 5-megabyte chunk of raw disk space, at an offset of 5200 kilobytes, to dbospace **dbspc3**.

```
onspaces -a dbspc3 \\.e: -o 5200 -s 5120
```

For reference information on adding a chunk to a dbospace with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

UNIX

Adding a Chunk with ON-Monitor

To add a chunk to a dbospace, follow these instructions:

1. Choose **Add Chunk→Dbspaces** option.
2. Use RETURN or the arrow keys to select the blobospace or dbospace that will receive the new chunk and press CTRL-B or F3.
3. The next screen indicates whether the blobospace or dbospace is mirrored. If it is, enter Y in the **Mirror** field.
4. If the dbospace to which you are adding the chunk is a temporary dbospace, enter Y in the **Temp** field.
5. If you indicated that the dbospace or blobospace is mirrored, you must specify both a primary chunk and mirrored chunk.

Enter the complete pathname for the new primary chunk in the **Full Pathname** field of the primary-chunk section.

6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in kilobytes, in the **Size** field.
8. If you are mirroring this chunk, enter the complete pathname, size, and optional offset in the mirror-chunk section of the screen.

Managing Blobspaces

This section discusses how to create a blobspace and determine the blobpage size. The database server stores TEXT and BYTE data in dbspaces or blobspaces, but blobspaces are more efficient. For information on adding a chunk, see [“Adding a Chunk to a Dbspace or Blobspace” on page 10-17](#).

Creating a Blobspace

You can use **onspaces**, ISA, or ON-Monitor to create a blobspace. Specify a blobspace name of up to 128 characters. The name must be unique and begin with a letter or underscore. You can use letters, digits, underscores, and \$ characters in the name.



Important: *You can mirror the blobspace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.*

Before you create a blobspace

1. Allocate disk space for the blobspace, as described in [“Allocating Disk Space” on page 10-6](#).
2. Determine what blobpage size is optimal for your environment.
For instructions, see [“Determining Blobpage Size” on page 10-21](#).

To create a blobspace using onspaces

1. To create a blobspace on UNIX, you must be logged in as user **informix** or **root**.
To create a blobspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is in online, quiescent mode, or the cleanup phase of fast-recovery mode.

3. To add a blobSpace, use the **onspaces -c -b** options.
 - a. Specify an explicit pathname for the blobSpace. If the blobSpace is mirrored, you must specify the pathname and size of both the primary chunk and mirrored chunk.
 - b. Use the **-o** option to specify an offset for the blobSpace.
 - c. Use the **-s** option to specify the size of the blobSpace chunk, in kilobytes.
 - d. Use the **-g** option to specify the blobpage size in terms of the number of disk pages per blobpages.

See [“Determining Blobpage Size” on page 10-21](#). For example, if your database server instance has a disk-page size of 2 kilobytes, and you want your blobpages to have a size of 10 kilobytes, enter 5 in this field.

If you specify an incorrect pathname, offset, or size, the database server does not create the blobSpace and displays an error message. Also see [“What to Do If You Run Out of Disk Space” on page 10-17](#).

4. After you create the blobSpace, you must perform a level-0 backup of the root dbSpace and the new blobSpace.

The following example creates a 10-megabyte mirrored blobSpace, **blobsp3**, with a blobpage size of 10 kilobytes, where the database server page size is 2 kilobytes. An offset of 200 kilobytes for the primary and mirrored chunks is specified. The blobSpace is created from raw disk space on UNIX.

```
onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10240 -m  
/dev/raw_dev2 200
```

For reference information on creating a blobSpace with onspaces, see the utilities chapter of the *Administrator's Reference*.

UNIX

To create a blobSpace with ON-Monitor

1. Select the **DbSpaces→BLOBSpace** option.
2. Enter the name of the new blobSpace in the **BLOBSpace Name** field.
3. If you want to create a mirror for the initial blobSpace chunk, enter Y in the **Mirror** field.
Otherwise, enter N.

4. Specify the blobpage size in terms of the number of disk pages per blobpage in the **BLOBPage Size** field.
See [“Determining Database Server Page Size” on page 10-22](#). For example, if your database server instance has a disk-page size of 2 kilobytes, and you want your blobpages to have a size of 10 kilobytes, enter 5 in this field.
5. Enter the complete pathname for the initial primary chunk of the blobspace in the **Full Pathname** field of the primary-chunk section.
6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in kilobytes, in the **Size** field.
8. If you are mirroring this blobspace, enter the full pathname, size, and optional offset in the mirror-chunk section of the screen.

Preparing Blobspaces to Store TEXT and BYTE Data

A newly created blobspace is not immediately available for storage of TEXT or BYTE data. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert TEXT and BYTE data into that blobspace appear in separate logical-log files. This requirement is true for all blobspaces, regardless of the logging status of the database. To accommodate this requirement, switch to the next logical-log file after you create a blobspace. (For instructions, see [“Backing Up Log Files to Free Blobpages” on page 13-12](#).)

Determining Blobpage Size

When you create a blobspace, use the size of the most frequently occurring simple large object as the size of the blobpage. In other words, choose a blobpage size that wastes the least amount of space. For information on calculating an optimal blobpage size, see blobpage size considerations in the chapter on effect of configuration on I/O activity in the *Performance Guide*.

If a table has more than one TEXT or BYTE column, and the objects are not close in size, store each column in a different blobspace, each with an appropriately sized blobpage. See [“Tables” on page 9-36](#).

Determining Database Server Page Size

When you specify blobpage size, you specify it in terms of the database server pages. You can use one of the following methods to determine the database server page size for your system:

- Run the **onstat -b** utility to display the system page size, given as buffer size on the last line of the output.
- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.
- In ON-Monitor, select either the **Parameters→Shared-Memory** or **Parameters→Initialize** option to display the system page size. ♦

Obtaining Blobspace Storage Statistics

To help you determine the optimal blobpage size for each blobspace, use the following database server utility commands:

- **oncheck -pe**
- **oncheck -pB**

The **oncheck -pe** command provides background information about the objects stored in a blobspace:

- Complete ownership information (displayed as *database:owner.table*) for each table that has data stored in the blobspace chunk
- The total number of pages used by each table to store its associated TEXT and BYTE data
- The total free and total overhead pages in the blobspace

The **oncheck -pB** command lists the following statistics for each table or database:

- The number of blobpages used by the table or database in each blobspace
- The average fullness of the blobpages used by each simple large object stored as part of the table or database

For more information, see [“Monitoring Blobspace Usage with oncheck -pe” on page 10-52](#), [“Determining Blobpage Fullness with oncheck -pB” on page 10-51](#), and **optimizing blobspace blobpage size** in the chapter on table performance considerations in the *Performance Guide*.

Managing Sbspaces

This section describes how to create a standard or temporary sbspace, monitor the metadata and user-data areas, add a chunk to an sbspace, and alter storage characteristics of smart large objects.

Creating an Sbspace

Use the onspaces utility or ISA to create an sbspace.

To create an sbspace using onspaces

1. To create an sbspace on UNIX, you must be logged in as user **informix** or **root**.
To create an sbspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is online, in quiescent mode, or in the cleanup phase of fast-recovery mode.

3. Use the **onspaces -c -S** options to create the sbspace.
 - a. Use the **-p** option to specify the pathname, the **-o** option to specify the offset, and the **-s** option to specify the sbspace size.
 - b. If you wish to mirror the sbspace, use the **-m** option to specify the mirror path and offset.
 - c. If you want to use the default storage characteristics for the sbspace, omit the **-Df** option.

If you want to specify different storage characteristics, use the **-Df** option. For more information, refer to [“Storage Characteristics of Sbspaces” on page 9-25](#).
 - d. The first chunk in an sbspace must have a metadata area.

You can specify a metadata area for an sbspace or let the database server calculate the size of the metadata area. For more information, see [“Sizing Sbspace Metadata” on page 10-25](#).
4. After you create the sbspace, you must perform a level-0 backup of the root dbspace and the new sbspace.
5. To start storing smart large objects in this sbspace, specify the space name in the SBSPACENAME configuration parameter.
6. Use **onstat -d**, **onstat -g smb s**, and **oncheck -cs, -cS, -ps, or -pS** to display information about the sbspace.

For more information, see [“Monitoring Sbspaces” on page 10-53](#).

This example creates a 20-megabyte mirrored sbspace, **sbsp4**. Offsets of 500 kilobytes for the primary and 500 kilobytes for the mirrored chunks are specified, as well as a metadata size of 150 kilobytes with a 200 kilobyte offset. The **AVG_LO_SIZE -Df** tag specifies an expected average smart-large-object size of 32 kilobytes.

```
onspaces -c -S sbsp4 -p /dev/rawdev1 -o 500 -s 20480 -m /dev/rawdev2 500  
-Ms 150 -Mo 200 -Df "AVG_LO_SIZE=32"
```

For information about creating an sbspace and default options for smart large objects, see **onspaces** in the utilities chapter of the *Administrator's Reference*. For information on creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQ/C Programmer's Manual*.

To create an sbspace using ISA

1. Create the sbspace using ISA. For more information, see the ISA online help.
2. Back up the new sbspace and the root dbspace.

Sizing Sbspace Metadata

The first chunk of an sbspace must have a metadata area. When you add smart large objects and chunks to the sbspace, the metadata area grows. In addition, the database server reserves 40 percent of the user area to be used in case the metadata area runs out of space.

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

- Let the database server calculate the size of the metadata area for the new sbspace chunk.
- Specify the size of the metadata area explicitly.

For instructions on estimating the size of the sbspace and metadata area, see table performance considerations in the *Performance Guide*. Also see [“Monitoring the Metadata and User-Data Areas” on page 10-59](#).

Adding a Chunk to an Sbspace

Use the **onspaces** utility or ISA to add a chunk to an sbspace or temporary sbspace. You can specify a metadata area for a chunk, let the database server calculate the metadata area, or use the chunk for user data only.

To add a chunk to an sbspace using onspaces

1. Ensure that the database server is online, in quiescent mode, or in the cleanup phase of fast-recovery mode.
2. Use the **onspaces -a** option to create the sbspace chunk.
 - a. Use the **-p** option to specify the pathname, the **-o** option to specify the offset, and the **-s** option to specify the chunk size.
 - b. If you want to mirror the chunk, use the **-m** option to specify the mirror path and offset.
 - c. To specify the size and offset of the metadata space, use the **-Mo** and **-Ms** options.

The database server allocates the specified amount of metadata area on the new chunk.
 - d. To allow the database server to calculate the size of the metadata for the new chunk, omit the **-Mo** and **-Ms** options.

The database server divides the estimated average size of the smart large objects by the size of the user data area.
 - e. To use the chunk for user data only, specify the **-U** option.

If you use the **-U** option, the database server does not allocate metadata space in this chunk. Instead, the sbspace uses the metadata area in one of the other chunks.
3. After you add a chunk to the sbspace, the database server writes the CHRESERV and CHKADJUP log records.
4. Perform a level-0 backup of the root dbspace and the sbspace.
5. Use **onstat -d** and **oncheck -pe** to monitor the amount of free space in the sbspace chunk.

For more information, see [“Monitoring Sbspaces” on page 10-53](#).

This example adds a 10-megabyte mirrored chunk to **sbsp4**. An offset of 200 kilobytes for both the primary and mirrored chunk is specified. If you are not adding a mirrored chunk, you can omit the **-m** option. The **-U** option specifies that the new chunk will contain user data exclusively.

```
onspaces -a sbsp4 -p /dev/rawdev1 -o 200 -s 10240 -m /dev/rawdev2 200 -U
```

For more information, see [“Adding a Chunk to a Dbspace or Blobspace” on page 10-17](#), and the onspaces section in the utilities chapter of the *Administrator’s Reference*.

Altering Storage Characteristics of Smart Large Objects

Use the **onspaces -ch** command to change the following default storage characteristics for the sbspace:

- Extent sizes
- Average smart-large-object size
- Buffering mode
- Last-access time
- Lock mode
- Logging

For more information, refer to [“Storage Characteristics of Sbspaces” on page 9-25](#) and managing sbspaces in the chapter on table performance considerations in your *Performance Guide*.

Creating a Temporary Sbspace

For background information and the rules for determining where temporary smart large objects are stored, see [“Temporary Sbspaces” on page 9-32](#). You can store temporary smart large objects in a standard or temporary sbspace. You can add or drop chunks in a temporary sbspace.

To create a temporary sbspace with a temporary smart large object

- 1. Allocate space for the temporary sbspace. For details, see “Allocating Disk Space” on page 10-6.

For information on SBSPACETEMP, see the configuration parameters chapter in the *Administrator’s Reference*.

- 2. Create the temporary sbspace as the following example shows:

```
onspaces -c -S tempsbsp -t -p ./tempsbsp -o 0 -s 1000
```

You can specify any of the following **onspaces** options:

- a. Specify a metadata area and offset (**-Ms** and **-Mo**).
- b. Specify storage characteristics (**-Df**).

You cannot turn on logging for a temporary sbspace.

- 3. Set the SBSPACETEMP configuration parameter to the name of the default temporary sbspace storage area.

Restart the database server.

- 4. Use **onstat -d** to display the temporary sbspace.

In [Figure 10-1](#), note the values of 0xa001 in the first **flags** column and NS in the second **flags** column for the temporary sbspace.

- 5. Specify the LO_CREATE_TEMP flag when you create a temporary smart large object.

Using DataBlade API:

```
mi_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
```

Using SQL/C:

```
ifx_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
```

For information on creating smart large objects, see the *IBM Informix DataBlade API Programmer’s Guide* and *IBM Informix SQL/C Programmer’s Manual*.

Figure 10-1
Output of onstat -d
Showing the
Temporary Sbspace

Dbspaces							
address	number	flags	fchunk	nchunks	flags	owner	name
ab01660	5	0xa001	5	1	NS	informix	tempsbsp
Chunks							
address	chk/dbs	offset	size	free	bpages	flags	pathname
ab01a50	5	5	0	500	347	347	POS ./tempsbsp
		Metadata	100	74	100		

Dropping a Chunk

Use **onspaces** or ISA to drop a chunk from a dbspace.

Before you drop a chunk, ensure that the database server is in the correct mode, using the following table as a guideline.

Chunk Type	Database Server in Online Mode	Database Server in Quiescent Mode	Database Server in Offline Mode
Dbspace chunk	Yes	Yes	No
Temporary dbspace chunk	Yes	Yes	No
Blobspace chunk	No	Yes	No
Sbspace or temporary sbspace chunk	Yes	Yes	No

Verifying Whether a Chunk Is Empty

To drop a chunk successfully from a dbspace with either of these utilities, the chunk must not contain any data. All pages other than overhead pages must be freed. If any pages remain allocated to nonoverhead entities, the utility returns the following error:

```
Chunk is not empty.
```

If this situation occurs, you must determine which table or other entity still occupies space in the chunk by executing **oncheck -pe** to list contents of the extent.

Usually, the pages can be removed when you drop the table that owns them. Then reenter the utility command.

Dropping a Chunk from a Dbspace with onspaces

The following example drops a chunk from **dbbsp3** on UNIX. An offset of 300 kilobytes is specified.

```
onspaces -d dbbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of a dbspace with the syntax in the previous example. Instead, you must drop the dbspace. Use the **fchunk** column of **onstat -d** to determine which is the initial chunk of a dbspace. For more information about **onstat**, see the utilities chapter of the *Administrator's Reference*.

For information about dropping a chunk from a dbspace with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

Dropping a Chunk from a Blobspace

The procedure for dropping a chunk from a blobspace is identical to the procedure for dropping a chunk from a dbspace described in [“Dropping a Chunk from a Blobspace with onspaces” on page 10-30](#) except that the database server must be in quiescent mode. Other than this condition, you need only substitute the name of your blobspace wherever a reference to a dbspace occurs.

Dropping a Chunk from an Sbspace with onspaces

The following example drops a chunk from **sbsp3** on UNIX. An offset of 300 kilobytes is specified. The database server must be in online or quiescent mode when you drop a chunk from an sbspace or temporary sbspace.

```
onspaces -d sbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of an sbspace with the syntax in the previous example. Instead, you must drop the sbspace. Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of an sbspace.

Using the -f (Force) Option

You can use the **-f** option of **onspaces** to drop an sbspace chunk without metadata allocated in it. If the chunk contains metadata for the sbspace, you must drop the entire sbspace. Use the **Chunks** section of **onstat -d** to determine which sbspace chunks contain metadata.

```
onspaces -d sbsp3 -f
```



Warning: *If you force the drop of an sbspace, you might introduce consistency problems between tables and sbspaces.*

Deleting Smart Large Objects Without any Pointers

Each smart large object has a reference count, the number of pointers to the smart large object. When the reference count is greater than 0, the database server assumes that the smart large object is in use and does not delete it.

Rarely, a smart large object with a reference count of 0 remains. You can use the **onspaces -cl** command to delete all smart large objects that have a reference count of 0, if it is not open by any application.

For information on using **onspaces -cl**, see the utilities chapter of the *Administrator's Reference*.

Dropping a Storage Space

Use **onspaces**, ISA, or ON-Monitor to drop a dbspace, temporary dbspace, blobspace, sbspace, temporary sbspace, or extspace.

On UNIX, you must be logged in as root or **informix** to drop a storage space. On Windows, you must be a member of the **Informix-Admin** group to drop a storage space.

You can drop a storage space only when the database server is in either online or quiescent mode.

Preparing to Drop a Storage Space

Before you drop a dbspace, you must first drop all databases and tables that you previously created in that dbspace. You cannot drop the root dbspace.

Before you drop a blobspace, you must drop all tables that have a TEXT or BYTE column that references the blobspace.

Execute **oncheck -pe** to verify that no tables or log files reside in the dbspace or blobspace.

Before you drop an sbspace, you must drop all tables that have a CLOB or BLOB column that reference objects that are stored in the sbspace. For sbspaces, you need not delete columns that point to an sbspace, but these columns must be null; that is, all smart large objects must be deallocated from the sbspace.



Tip: *If you drop tables on dbspaces where light appends are occurring, the light appends might be slower than you expect. The symptom of this problem is physical logging activity. If light appends are slower than you expect, make sure that no tables are dropped in the dbspace either before or during the light appends. If you have dropped tables, force a checkpoint with **onmode -c** before you perform the light append.*

Dropping a Mirrored Storage Space

If you drop a storage space that is mirrored, the mirror spaces are also dropped.

If you want to drop only a storage-space mirror, turn off mirroring. (See [“Ending Mirroring” on page 18-11](#).) This action drops the dbspace, blobspace, or sbspace mirrors and frees the chunks for other uses.

Dropping a Storage Space with onspaces

To drop a storage space with onspaces, use the **-d** option as illustrated in the following examples.

This example drops a dbspace called **dbspce5** and its mirrors.

```
onspaces -d dbspce5
```


This example drops a dbospace called **blobbsp3** and its mirrors.

```
onspaces -d blobbsp3
```

Use the **-d** option with the **-f** option if you want to drop an sbospace that contains data. If you omit the **-f** option, you cannot drop an sbospace that contains data. This example drops an sbospace called **sbspc4** and its mirrors.

```
onspaces -d sbspc4 -f
```

Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the deleted smart large objects.

For information on dropping a storage space with **onspaces**, see the utilities chapter of the *Administrator's Reference*.



UNIX

Dropping a Dbospace or Blobospace with ON-Monitor

To drop a dbospace or blobospace with ON-Monitor, follow these instructions:

1. Select the **Dbspaces→Drop** option.
2. Use RETURN or arrow keys to scroll to the dbospace or blobospace that you want to drop.
3. Press CTRL-B or F3.

You are asked to confirm that you want to drop the dbospace or blobospace.

Backing Up After Dropping a Storage Space

If you create a storage space with the same name as the deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see the *IBM Informix Backup and Restore Guide*.

Warning: After you drop a dbospace, blobospace, or sbospace, the newly freed chunks are available for reassignment to other dbspaces, blobspaces, or sbspaces. However, before you reassign the newly freed chunks, you must perform a level-0 backup of the root dbospace and the modified storage space. If you do not perform this backup, and you subsequently need to perform a restore, the restore might fail because the backup reserved pages are not up-to-date.



Managing Extspaces

An extspace does not require allocation of disk space. You create and drop extspaces using the **onspaces** utility. For more information about extspaces, refer to [“Extspaces” on page 9-35](#).

Creating an Extspace

You create an extspace with the **onspaces** utility. But you should first have a valid datasource and a valid access method with which to access that data source. Although you can create an extspace without a valid access method or a valid datasource, any attempts to retrieve data from the extspace will generate an error. For information on access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.

To create an extspace with **onspaces**, use the **-c** option as illustrated in the following example. This example creates an extspace, **pass_space**, that is associated with the UNIX password file.

```
onspaces -c -x pass_space -l /etc/passwd
```

Specify an extspace name of up to 128 characters. The name must be unique and begin with a letter or underscore. You can use letters, digits, underscores, and \$ characters in the name.



Important: *The preceding example assumes that you have coded a routine that provides functions for correctly accessing the file **passwd** and that the file itself exists. Once you have created the extspace, you must use the appropriate commands to allow access to the data in the file **passwd**. For more information on user-defined access methods, see the “IBM Informix Virtual-Table Interface Programmer's Guide.”*

For reference information on creating an extspace with **onspaces**, see the utilities chapter in the *Administrator's Reference*.

Dropping an Extspace

To drop an extspace with onspaces, use the **-d** option as illustrated in the following examples. An extspace cannot be dropped if it is associated with an existing table or index.

This example drops an extspace called **pass_space**.

```
onspaces -d pass_space
```

Skipping Inaccessible Fragments

One benefit that fragmentation provides is the ability to skip table fragments that are unavailable during an I/O operation. For example, a query can proceed even when a fragment is located on a chunk that is currently down as a result of a disk failure. When this situation occurs, a disk failure affects only a portion of the data in the fragmented table. By contrast, tables that are not fragmented can become completely inaccessible if they are located on a disk that fails.

This functionality is controlled as follows:

- By the database server administrator with the DATASKIP configuration parameter
- By individual applications with the SET DATASKIP statement

Using the DATASKIP Configuration Parameter

You can set the DATASKIP parameter to OFF, ALL, or ON *dbspace_list*. OFF means that the database server does not skip any fragments. If a fragment is unavailable, the query returns an error. ALL indicates that any unavailable fragment is skipped. ON *dbspace_list* instructs the database server to skip any fragments that are located in the specified dbspaces.

Using the Dataskip Feature of onspaces

Use the **dataskip** feature of the **onspaces** utility to specify the dbspaces that are to be skipped when they are unavailable. For example, the following command sets the **DATASKIP** parameter so that the database server skips the fragments in **dbspace1** and **dbspace3**, but not in **dbspace2**:

```
onspaces -f ON dbspace1 dbspace3
```

For the complete syntax of this **onspaces** option, see the utilities chapter in the *Administrator's Reference*.

Using onstat to Check Dataskip Status

Use the **onstat** utility to list the dbspaces currently affected by the **dataskip** feature. The **-f** option lists both the dbspaces that were set with the **DATASKIP** configuration parameter and the **-f** option of the **onspaces** utility. When you execute **onstat -f**, you see one of the following messages:

```
dataskip is OFF for all dbspaces
```

```
dataskip is ON for all dbspaces
```

```
dataskip is ON for dbspaces:  
dbspace1 dbspace2 ...
```

Using the SQL Statement SET DATASKIP

An application can use the SQL statement **SET DATASKIP** to control whether a fragment should be skipped if it is unavailable. Applications should include this statement only in limited circumstances, because it causes queries to return different results, depending on the availability of the underlying fragments. Like the configuration parameter **DATASKIP**, the **SET DATASKIP** statement accepts a list of dbspaces that indicate to the database server which fragments to skip. For example, suppose that an application programmer included the following statement at the beginning of an application:

```
SET DATASKIP ON dbspace1, dbspace5
```

This statement causes the database server to skip **dbspace1** or **dbspace5** whenever both of these conditions are met:

- The application attempts to access one of the dbspaces.
- The database server finds that one of the dbspaces is unavailable.

If the database server finds that both **dbspace1** and **dbspace5** are unavailable, it skips both dbspaces.

The DEFAULT setting for the SET DATASKIP statement allows a database server administrator to control the dataskip feature. Suppose that an application developer includes the following statement in an application:

```
SET DATASKIP DEFAULT
```

When a query is executed subsequent to this SQL statement, the database server checks the value of the configuration parameter DATASKIP. Encouraging end users to use this setting allows the database server administrator to specify which dbspaces are to be skipped as soon as the database server administrator becomes aware that one or more dbspaces are unavailable.

Effect of the Dataskip Feature on Transactions

If you turn the dataskip feature on, a SELECT statement always executes. In addition, an INSERT statement always succeeds if the table is fragmented by round-robin and at least one fragment is online. However, the database server *does not* complete operations that write to the database if a possibility exists that such operations might compromise the integrity of the database. The following operations fail:

- All UPDATE and DELETE operations where the database server cannot eliminate the down fragments
If the database server *can* eliminate the down fragments, the update or delete is successful, but this outcome is independent of the DATASKIP setting.
- An INSERT operation for a table fragmented according to an expression-based distribution scheme where the appropriate fragment is down

- Any operation that involves referential constraint checking if the constraint involves data in a down fragment
For example, if an application deletes a row that has child rows, the child rows must also be available for deletion.
- Any operation that affects an index value (for example, updates to a column that is indexed) where the index in question is located in a down chunk

Determining When to Use Dataskip

Use this feature sparingly and with caution because the results are always suspect. Consider using it in the following situations:

- You can accept the compromised integrity of transactions.
- You can determine that the integrity of the transaction is not compromised.

The latter task can be difficult and time consuming.

Determining When to Skip Selected Fragments

In certain circumstances, you might want the database server to skip some fragments, but not others. This usually occurs in the following situations:

- Fragments can be skipped because they do not contribute significantly to a query result.
- Certain fragments are down, and you decide that skipping these fragments and returning a limited amount of data is preferable to canceling a query.

When you want to skip fragments, use the ON *dbspace-list* setting to specify a list of dbspaces with the fragments that the database server should skip.

Determining When to Skip All Fragments

Setting the DATASKIP configuration parameter to ALL causes the database server to skip all unavailable fragments. Use this option with caution. If a dbspace becomes unavailable, all queries initiated by applications that do not issue a SET DATASKIP OFF statement before they execute could be subject to errors.

Monitoring Fragmentation Use

The database administrator might find the following aspects of fragmentation useful to monitor:

- Data distribution over fragments
- I/O request balancing over fragments
- The status of chunks that contain fragments

The administrator can monitor the distribution of data over table fragments. If the goal of fragmentation is improved single-user response time, it is important for data to be distributed evenly over the fragments. To monitor fragmentation disk use, you must monitor database server tblspaces, because the unit of disk storage for a fragment is a tblspace. (For information on how to monitor the data distribution for a fragmented table, see [“Monitoring Tblspaces and Extents” on page 10-48.](#))

The administrator must monitor I/O request queues for data that is contained in fragments. When I/O queues become unbalanced, the administrator should work with the DBA to tune the fragmentation strategy. (For a discussion of how to monitor chunk use, including the I/O queues for each chunk, see [“Monitoring Chunks” on page 10-41.](#))

The administrator must monitor fragments for availability and take appropriate steps when a dbspace that contains one or more fragments fails. For how to determine if a chunk is down, see [“Monitoring Chunks” on page 10-41.](#)

Displaying Databases

You can display databases that you create with the following tools:

- SMI tables
- ISA
- ON-Monitor

Using SMI Tables

Query the **sysdatabases** table to display a row for each database managed by the database server. For a description of the columns in this table, see the **sysdatabases** information in the chapter about the **sysmaster** database in the *Administrator's Reference*.

Using ISA

To query **sysdatabases** using ISA, follow these steps:

1. Choose **SQL→Query**.
2. Select the **sysmaster** data in the **Database** list.
3. Enter the following command and click **Submit**:

```
select * from sysdatabases;
```

Using ON-Monitor

To use ON-Monitor to find the current status of each database, select the **Status→Databases** option. ON-Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in the previous section.

Monitoring Disk Usage

This section describes methods of tracking the disk space used by various database server storage units.

For background information about internal database server storage units mentioned in this section, see the chapter about disk structures and storage in the *Administrator's Reference*.

Monitoring Chunks

You can monitor chunks for the following information:

- Chunk size
- Number of free pages
- Tables within the chunk

This information allows you to track the disk space used by chunks, monitor chunk I/O activity, and check for fragmentation.

onstat -d

The **onstat -d** utility lists all dbspaces, blobspaces, and sbspaces and the following information for the chunks within those spaces.

- The address of the chunk
- The chunk number and associated dbspace number
- The offset into the device (in pages)
- The size of the chunk (in pages)
- The number of free pages in the chunk
- The pathname of the physical device

If you issue the **onstat -d** command on an instance with blobspace chunks, the number of free pages shown is out of date. The tilde (~) that precedes the free value indicates that this number is approximate. The **onstat -d** command does not register a blobpage as available until the logical log in which a deletion occurred is backed up and the blobpage is freed. Therefore, if you delete 25 simple large objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

To obtain an accurate number of free blobpages in a blobspace chunk, issue the **onstat -d update** command. For details, see “[onstat -d update](#)” on [page 10-43](#).

[Figure 10-2](#) shows sample **onstat -d** output. The **flags** column in the **chunk** section provides the following information:

- Whether the chunk is the primary chunk or the mirrored chunk
- Whether the chunk is online, is down, is being recovered, or is a new chunk

Important: You must perform a level-0 backup of the root dbspace and the modified dbspace before mirroring can become active and after turning off mirroring.



```
Dbspaces
address  number  flags    fchunk  nchunks  flags    owner   name
40c980   1         0x1      1        1         N       informix rootdbs
40c9c4   2         0x1      2        1         N       informix fstdbs
40ca08   3         0x11     3        1         N B     informix fstblob
3 active, 2047 maximum

Note: For BLOB chunks, the number of free pages shown is out of date. Run
'onstat -d update' for current stats.

Chunks
address  chk/dbs  offset  size   free   bpages  flags  pathname
40c224   1 1 0     20000  14001          PO-   /home/server/root_chunk
40c2bc   2 2 0     2000   1659          PO-   /home/server/fst_chunk
40c354   3 3 0    12500 ~6250   6250      POB   /home/server/blob_chunk
3 active, 2047 maximum
```

Figure 10-2
onstat -d Output

onstat -d update

The **onstat -d update** option displays the same information as **onstat -d** and an accurate number of free blobpages for each blobspace chunk. In [Figure 10-3](#), a blobspace called **fstblob** contains a single chunk called **blob_chunk**.

```

Dbspaces
address  number  flags  fchunk  nchunks  flags  owner  name
a7317d8  1      0x1   1       1       N      informix rootdbs
40c9c4   2      0x1   2       1       N      informix fstdbs
40ca08   3      0x11  3       1       N B    informix fstblob
3 active, 2047 maximum

Waiting for server to update BLOB chunk statistics:

Chunks
address  chk/dbs  offset  size  free  bpages  flags  pathname
40c224   1 1 0    20000 14001  PO-    /home/server/root_chunk
40c2bc   2 2 0    2000  1659  PO-    /home/server/fst_chunk
40c354   3 3 0    12500 ~6237  POB    /home/server/blob_chunk
3 active, 2047 maximum

```

Figure 10-3
onstat -d update
Output with
Information
About
Blobspaces

onstat -D

The **onstat -D** option displays the same information as **onstat -d**, plus the number of pages read from the chunk (in the **page Rd** field).

[Figure 10-4](#) shows sample output.

```

Dbspaces
address  number  flags  fchunk  nchunks  flags  owner  name
40d100   1      1      1       1       N      informix rootdbs
40d144   2      2      2       1       M      informix cookedspace
40d188   3      10     3       1       N B    informix cookedblob
3 active, 2047 maximum

Chunks
address  chk/dbs  offset  page Rd  page Wr  pathname
40c274   1 1 0    146     4      /home/server/root_chunk
40c30c   2 2 0     1     0      /home/server/test_chunk
40c8fc   2 2 0    36     0      /home/server/test_mirr
40c3a4   3 3 0     4     0      /home/server/blob_chunk
3 active, 2047 maximum

```

Figure 10-4
onstat -D Output

onstat -g iof

The **onstat -g iof** option displays the number of reads from each chunk and the number of writes to each chunk. If one chunk has a disproportionate amount of I/O activity against it, this chunk might be a system bottleneck. This option is useful for monitoring the distribution of I/O requests against the different fragments of a fragmented table. [Figure 10-5](#) shows sample output.

```
...
AIO global files:
gfd pathname          totalops  dskread dskwrite  io/s
 3 raw_chunk          38808      27241  11567    6.7
 4 cooked_chk1        7925       5660   2265    1.4
 5 cooked_chk2        3729       2622   1107    0.6
```

Figure 10-5
onstat -g iof Output

oncheck -pr

The database server stores chunk information in the reserved pages PAGE_1PCHUNK and PAGE_2PCHUNK.

To list the contents of the reserve pages, execute **oncheck -pr**. [Figure 10-6](#) shows sample output for **oncheck -pr**. This output is essentially the same as the **onstat -d** output; however, if the chunk information has changed since the last checkpoint, these changes do not appear in the **oncheck -pr** output.

```
Validating PAGE_1DBSP & PAGE_2DBSP...
    Using dbspace page PAGE_2DBSP.

    DBspace number      1
    DBspace name        rootdbs
    Flags               0x20001          No mirror chunks
    Number of chunks     2
    First chunk         1
    Date/Time created   07/28/2000 14:46:55
    Partition table page number 14
    Logical Log Unique Id 0
    Logical Log Position 0
    Oldest Logical Log Unique Id 0
    Last Logical Log Unique Id 0
    DBspace archive status No archives have occurred
.
.
Validating PAGE_1PCHUNK & PAGE_2PCHUNK...
    Using primary chunk page PAGE_2PCHUNK.

    Chunk number        1
    Flags               0x40          Chunk is online
    Chunk path          /home/server/root_chunk
    Chunk offset        0 (p)
    Chunk size          75000 (p)
    Number of free pages 40502
    DBSpace number      1
.
.
.
```

Figure 10-6
oncheck -pr Output
Showing Dbspace
and Chunk
Information

oncheck -pe

To obtain the physical layout of information in the chunk, execute **oncheck -pe**. The dbspaces, blobspaces, and sbspaces are listed. [Figure 10-7 on page 10-46](#) shows sample output for **oncheck -pe**.

The following information is displayed:

- The name, owner, and creation date of the dbspace
- The size in pages of the chunk, the number of pages used, and the number of pages free
- A listing of all the tables in the chunk, with the initial page number and the length of the table in pages

The tables within a chunk are listed sequentially. This output is useful for determining chunk fragmentation. If the database server is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

```
DBSpace Usage Report:  rootdbs                Owner:  informix  Created: 08/08/2000

Chunk  Pathname                Size      Used      Free
   1    /home/server/root_chunk      75000    19420    55580

Description                Offset      Size
-----
RESERVED PAGES                0           12
CHUNK FREELIST PAGE           12           1
rootdbs:'informix'.TBLSpace    13          250
PHYSICAL LOG                  263         1000
FREE                          1263        1500
LOGICAL LOG: Log file 2        2763        1500
LOGICAL LOG: Log file 3        4263        1500
...
sysmaster:'informix'.sysdatabases 10263         4
sysmaster:'informix'.systables    10267         8
...

Chunk  Pathname                Size      Used      Free
   2    /home/server/dbspace1      5000         53    4947

Description                Offset      Size
-----
RESERVED PAGES                0           2
CHUNK FREELIST PAGE           2           1
dbspace1:'informix'.TBLSpace     3           50
FREE                          53         4947
```

Figure 10-7
oncheck -pe Output

Using IBM Informix Server Administrator

You can perform the following tasks using ISA commands:

- Check reserved pages.
- Check storage spaces.
- Add dbspaces, temporary dbspaces, blobspaces, temporary sbspaces, and sbspaces.
- Display and add chunks to a storage space.
- Check the dataskip status.
- Display and add external spaces.

UNIX

- Display the number of pages in your database, percentage of allocated space, and used space.
- Override ONDBSPACEDOWN.

Using ON-Monitor

You can perform the following tasks using ON-Monitor commands.

ON_Monitor Command	Description
Status→Spaces	Displays status information about storage spaces and chunks
Dbspaces→Create	Creates a dbspace
Dbspaces→BLOBSpace	Creates a blobspace
Dbspaces→Mirror	Adds or drops mirroring for a storage space
Dbspaces→Info	Displays information about storage spaces
Dbspaces→Add Chunk	Adds a chunk to a storage space
Dbspaces→dataSkip	Starts or stops dataskip
Dbspaces→Chunk	Adds a chunk to a dbspace or blobspace
Dbspaces→Drop	Drops a dbspace or blobspace
Dbspaces→Status	Changes the mirror status of a chunk

Using SMI Tables

Query the **syschunks** table to obtain the status of a chunk. The following columns are relevant.

Column	Description
chknum	Number of the chunk within the dbspace
dbnum	Number of the dbspace
chksize	Total size of the chunk in pages
nfree	Number of pages that are free
is_offline	Whether the chunk is down
is_recovering	Whether the chunk is recovering
mis_offline	Whether the mirrored chunk is down
mis_recovering	Whether the mirrored chunk is being recovered

The **syschkio** table contains the following columns.

Column	Description
pagesread	Number of pages read from the chunk
pageswritten	Number of pages written to the chunk

Monitoring Tblspaces and Extents

Monitor tblspaces and extents to determine disk usage by database, table, or table fragment. Monitoring disk usage by table is particularly important when you are using table fragmentation, and you want to ensure that table data and table index data are distributed appropriately over the fragments.

Execute **oncheck -pt** to obtain extent information. The **oncheck -pT** option returns all the information from the **oncheck -pt** option as well as the additional information about page and index usage.

Using SMI Tables

Query the **systabnames** table to obtain information about each tblspace. The **systabnames** table has columns that indicate the corresponding table, database, and table owner for each tblspace.

Query the **sysextents** table to obtain information about each extent. The **sysextents** table has columns that indicate the database and the table that the extent belongs to, as well as the physical address and size of the extent.

Monitoring Simple Large Objects in a BlobSpace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

onstat -O

The **onstat -O** option displays information about the staging-area blobspace and the Optical Subsystem memory cache. [Figure 10-8](#) shows an example of the output from this option. The totals shown in the display accumulate from session to session. The database server resets the totals to 0 only when you execute **onstat -z**.

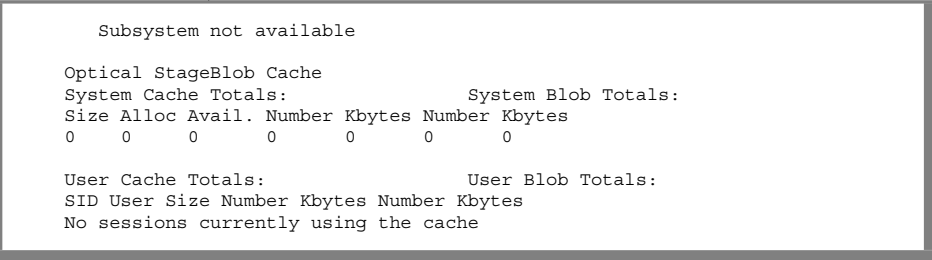


Figure 10-8
onstat -O Output

The first section of the display describes the following system-cache totals information.

Column	Description
size	Size specified in the OPCACHEMAX configuration parameter
alloc	Number of 1-kilobyte pieces that the database server allocated to the cache
avail	Portion of alloc (in kilobytes) not used
number	Number of simple large objects that the database server successfully put into the cache without overflowing
kbytes	Number of kilobytes of the simple large objects that the database server put into the cache without overflowing
number	Number of simple large objects that the database server wrote to the staging-area blobSpace
kbytes	Number of kilobytes of simple large objects that the database server wrote to the staging-area blobSpace

Although the **size** output indicates the amount of memory that is specified in the configuration parameter OPCACHEMAX, the database server does not allocate memory to OPCACHEMAX until necessary. Therefore, the **alloc** output reflects only the number of 1-kilobyte pieces of the largest simple large object that has been processed. When the values in the **alloc** and **avail** output are equal, the cache is empty.

The second section of the display describes the following user-cache totals information.

Column	Description
SID	Session ID for the user
user	User ID of the client
size	Size specified in the INFORMIXOPCACHE environment variable, if set If you do not set the INFORMIXOPCACHE environment variable, the database server uses the size that you specify in the configuration parameter OPCACHEMAX .
number	Number of simple large objects that the database server put into cache without overflowing
kbytes	Number of kilobytes of simple large objects that the database server put into the cache without overflowing
number	Number of simple large objects that the database server wrote to the staging-area blobpage
kbytes	Size of the simple large objects (in kilobytes) that the database server wrote to the staging-area blobpage

Determining Blobpage Fullness with oncheck -pB

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobpage.

Execute **oncheck -pB** with either a database name or a table name as a parameter. The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores_demo** database:

```
oncheck -pB stores_demo:sriram.catalog
```

For detailed information about interpreting the **oncheck -pB** output, see optimizing blobpage size in the chapter on table performance considerations in the *Performance Guide*.

Monitoring BlobSpace Usage with oncheck -pe

The **oncheck -pe** command provides information about blobSpace usage:

- Names of the tables that store TEXT and BYTE data, by chunk
- Number of disk pages (*not* blobpages) used, by table
- Number of free disk pages remaining, by chunk
- Number of overhead pages used, by chunk

Figure 10-9 shows sample **oncheck -pe** output.

```
BLOBSpace Usage Report:  fstblob          Owner:  informix  Created: 03/01/99
Chunk: 3    /home/server/blob_chunk      Size    Used    Free
                                     4000    304    3696

Disk usage for Chunk 3              Total Pages
-----
OVERHEAD                             8
stores_demo:chrisw.catalog          296
FREE                                3696
```

Figure 10-9
oncheck -pe
Output That
Shows BlobSpace
Use

Monitoring Simple Large Objects in a DbSpace with oncheck -pT

Use **oncheck -pT** to monitor dbSpaces to determine the number of dbSpace pages that TEXT and BYTE data use.

This command takes a database name or a table name as a parameter. For each table in the database, or for the specified table, the database server displays a general tblspace report.

Following the general report is a detailed breakdown of page use in the extent, by page type. See the **Type** column for information on TEXT and BYTE data.

The database server can store more than one simple large object on the same blobpage. Therefore, you can count the number of pages that store TEXT or BYTE data in the tblspace, but you cannot estimate the number of simple large objects in the table.

Figure 10-10 shows sample output.

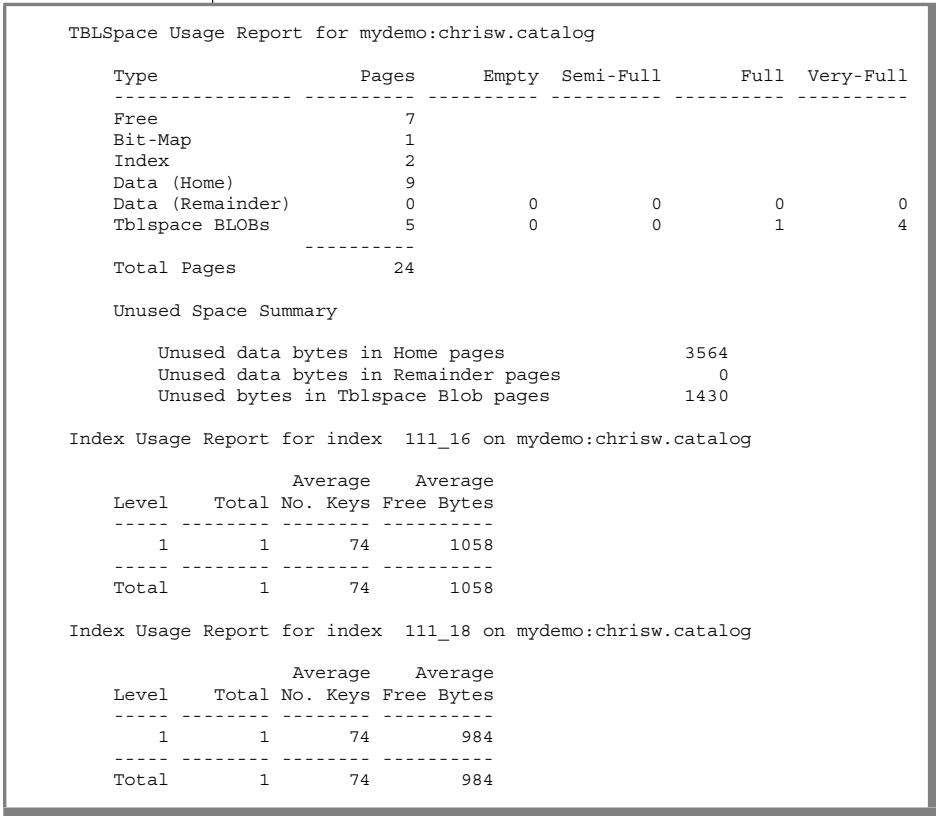


Figure 10-10
Sample output from
oncheck -pT with
TEXT or BYTE data

Monitoring Sbspaces

One of the most important areas to monitor in an sbspace is the metadata page use. When you create an sbspace, you specify the size of the metadata area. Also, any time that you add a chunk to the sbspace, you can specify that metadata space be added to the chunk.

If you attempt to insert a new smart large object, but no metadata space is available, you receive an error. The administrator should monitor metadata space availability to prevent this situation from occurring.

Use the following commands to monitor sbspaces.

Command	Description
onstat -g smb s	Displays the storage attributes for all sbspaces in the system: <ul style="list-style-type: none">■ sbspace name, flags, owner■ logging status■ average smart-large-object size■ first extent size, next extent size, and minimum extent size■ maximum I/O access time■ lock mode
onstat -g smb c	Displays for each sbspace chunk the following: <ul style="list-style-type: none">■ chunk number and sbspace name■ chunk size and pathname■ total user data pages and free user data pages■ location and number of pages in each user-data and metadata areas See “Using onstat -g smb c” on page 10-60.
oncheck -ce oncheck -pe	Displays the following information about sbspace use: <ul style="list-style-type: none">■ Names of the tables that store smart-large-object data, by chunk■ Number of disk pages (not sbpages) used, by table■ Number of free user-data pages that remain, by chunk■ Number of reserved user-data pages that remain, by chunk■ Number of metadata pages used, by chunk The output provides the following totals: <ul style="list-style-type: none">■ Total number of used pages for all user-data areas and metadata area. The system adds 53 pages for the reserved area to the totals for the user-data area and metadata area.■ Number of free pages that remain in the metadata area■ Number of free pages that remain in all user-data areas See “Using oncheck -ce and oncheck -pe” on page 10-56 and “Monitoring the Metadata and User-Data Areas” on page 10-59.

Command	Description
onstat -d	Displays the following information about the chunks in each sbspace: <ul style="list-style-type: none">■ Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data areas■ Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data areas See “Using onstat -d,” following.
oncheck -cs oncheck -ps	Validates and displays information about the metadata areas for sbspaces. See “Using oncheck -cs” on page 10-57 and “Using oncheck -ps” on page 10-58.
oncheck -cS	Displays information about smart-large-object extents and user-data areas for sbspaces.
oncheck -pS	Displays information about smart-large-object extents, user-data areas, and metadata areas for sbspaces. For more information on oncheck -cS and -pS , see managing sbspaces in the chapter on table performance considerations in your <i>Performance Guide</i> .

(2 of 2)

Using onstat -d

Use the **onstat -d** option to display the following information about the chunks in each sbspace:

- Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data area
- Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data area

In [Figure 10-11](#), the **onstat -d** output displays information about the **rootdbs** and the sbspace **s9_sbsp**. The **8000** and **s** flags in the **Flags** columns indicate an sbspace. For each sbspace chunk, the first row displays information about the whole sbspace and user-data area. The second row displays information about the metadata area. In this example, the **Size** column shows a total of 1000 pages for **s9_sbsp**. The user-data area is 842 pages with 726 pages free. The metadata area is 105 pages with 60 pages free.

To find out the total amount of used space, execute the **oncheck -pe** command. For more information, see [“Using oncheck -ce and oncheck -pe” on page 10-56](#).

```
Dbspaces
address number flags fchunk nchunks flags owner name
alb01d8 1 1 1 1 N informix rootdbs
alb0658 2 8000 2 1 N S informix s9_sbspc
2 active, 2047 maximum

Chunks
address chk/dbs offset size free bpages flags pathname
alb0320 1 1 0 75000 64588 PO-
/ix/ids9.2/root_chunk
alb04f8 2 2 0 1000 726 842 POS
/ix/ids9.2/./s9_sbspc
Metadata 105 60 105
2 active, 2047 maximum
```

Figure 10-11
*onstat -d Output
with Information
About Sbspaces*

The **onstat -d** option does not register an sbpage as available until the logical log in which a deletion occurred is backed up and the sbpage is freed. Therefore, if you delete 25 smart large objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

Using oncheck -ce and oncheck -pe

Execute **oncheck -ce** to display the size of each sbspace chunk, the total amount of used space, and the amount of free space in the user-data area. The **oncheck -pe** option displays the same information as **oncheck -ce** plus a detailed listing of chunk use. First the dbspaces are listed and then the sbspaces. The **-pe** output provides the following information about sbspace use:

- Names of the tables that store smart-large-object data, by chunk
- Number of disk pages (*not* sbpages) used, by table
- Number of free user-data pages that remain, by chunk
- Number of metadata pages used, by chunk

The output provides the following totals:

- Total number of used pages for the user-data area, metadata area, and reserved area
- The system adds an extra 53 pages for the reserved area to the totals for the user-data area and metadata area.
- Number of free pages that remain in the metadata area
- Number of free pages that remain in the user-data area



Tip: The *oncheck -pe* option provides information about sbspace use in terms of database server pages, not sbpages.

Figure 10-12 shows sample output. In this example, the sbspace **s9_sbspc** has a total of 214 used pages, 60 free pages in the metadata area, and 726 free pages in the user-data area.

Chunk Pathname	Size	Used	Free
2 /ix/ids9.2/./s9_sbspc	1000	940	60
Description	Offset	Size	
RESERVED PAGES	0	2	
CHUNK FREELIST PAGE	2	1	
s9_sbspc:'informix'.TBLSpace	3	50	
SBLOBSpace LO [2,2,1]	53	8	
SBLOBSpace LO [2,2,2]	61	1	
...			
SBLOBSpace LO [2,2,79]	168	1	
SBLOBSpace FREE USER DATA	169	305	
s9_sbspc:'informix'.sbspace_desc	474	4	
s9_sbspc:'informix'.chunk_adjunc	478	4	
s9_sbspc:'informix'.LO_hdr_partn	482	8	
s9_sbspc:'informix'.LO_ud_free	490	5	
s9_sbspc:'informix'.LO_hdr_partn	495	24	
FREE	519	60	
SBLOBSpace FREE USER DATA	579	421	
Total Used:	214		
Total SBLOBSpace FREE META DATA:	60		
Total SBLOBSpace FREE USER DATA:	726		

Figure 10-12
oncheck -pe Output
That Shows
Sbspace Use

Using oncheck -cs

The **oncheck -cs** and the **oncheck -Cs** options validate the metadata area of an sbspace. Figure 10-13 shows an example of the **-cs** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Use the **oncheck -cs** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area. To find the number of used pages in the metadata area, total the numbers in the **Used** column. To find the number of free pages in the metadata area, total the numbers in the **Free** column.

For example, based on the field values displayed in [Figure 10-13](#), the total number of used pages in the metadata area for **s9_sbspc** is 33 two-kilobyte pages (or 66 kilobytes). The metadata area contains a total of 62 free pages (or 124 kilobytes).

```
Validating space 's9_sbspc' ...
```

SBLIOspace Metadata Partition	Partnum	Used	Free
s9_sbspc:'informix'.TBLSpace	0x200001	6	44
s9_sbspc:'informix'.sbspace_desc	0x200002	2	2
s9_sbspc:'informix'.chunk_adjunc	0x200003	2	2
s9_sbspc:'informix'.LO_hdr_partn	0x200004	21	11
s9_sbspc:'informix'.LO_ud_free	0x200005	2	3

Figure 10-13
oncheck -cs Output

Using oncheck -ps

The **oncheck -ps** option validates and displays information about the metadata areas in sbspace partitions. [Figure 10-14 on page 10-59](#) shows an example of the **-ps** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** validates and displays tblspace information for all storage spaces.

To monitor the amount of free metadata space, run the following command:

```
oncheck -ps spacename
```

The **-ps** output includes information about the locking granularity, **partnum**, number of pages allocated and used, extent size, and number of rows in the metadata area. Use the **oncheck -ps** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area.

If you run **oncheck -ps** for the dbspace that contains the tables where the smart large objects are stored, you can find the number of rows in the table.

Figure 10-14
oncheck -ps Output

```
Validating space 's9_sbspc' ...

TBLSpace Report for
TBLspace Flags                2801      Page Locking
                                TBLSpace use 4 bit bit-maps
                                Permanent System TBLSpace

Partition partnum              0x200001
Number of rows                  92
Number of special columns      0
Number of keys                  0
Number of extents              1
Current serial value            1
First extent size               50
Next extent size               50
Number of pages allocated      50
Number of pages used            6
Number of data pages           0
Number of rows                  0
Partition lockid               2097153
Optical Cluster Partnum        -1
Current SERIAL8 value           1
Current REFID value             1
Created                        Thu Jun 25 14:14:33 1999
```

Monitoring the Metadata and User-Data Areas

The database server reserves 40 percent of the user-data area as a *reserved area*. The database server uses this reserved space for either the metadata or user data. The metadata area gets used up as smart large objects are added to that sbspace. When the database server runs out of metadata or user-data space, it moves a block of the reserved space to the corresponding area.

When all of the reserve area is used up, the database server cannot move space to the metadata area, even if the user-data area contains free space.

- 1. As you add smart large objects to the sbspace, use **oncheck -pe** or **onstat -g smb c** to monitor the space in the metadata area, user-data area, and reserved area. For an example, see [“Using oncheck -ce and oncheck -pe” on page 10-56](#).
- 2. Use the message log to monitor metadata stealing.
The database server prints messages about the number of pages allocated from the reserved area to the metadata area.

3. Add another chunk to the sbspace before the sbspace runs out of space in the metadata and reserved areas.
For more information, see [“Adding a Chunk to an Sbspace” on page 10-26](#).
4. The database server writes the FREE_RE and CHKADJUP log records when it moves space from the reserve area to the metadata or user-data area.

For more information, see [“Sizing Sbspace Metadata” on page 10-25](#).

Using `onstat -g smb c`

Use the `onstat -g smb c` option to monitor the amount of free space in each sbspace chunk, and the size in pages of the user-data, metadata, and reserved areas. In [Figure 10-15](#), chunk 2 of **sbspace1** has 2253 used pages (**usr pgs**) and 2245 free pages (**free pg**). For the first user-data area **Ud1**, the starting page offset is 53 and the number of pages is 1126. For the metadata area **Md**, the starting page offset is 1179 and the number of pages is 194. For the reserved data **Ud2**, the starting page offset is 1373 and the number of pages is 1127.

```
Chunk Summary:

sbnnum 2  chunk 2
chunk:    address  flags   offset  size  orig fr  usr pgs  free pg
          303cf2a8  F-----  0      2500   2253    2253    2245
          path: /usr11/myname/sbspace1

          start pg  npages
Ud1   :    53      1126
Md    :   1179     194
Ud2   :   1373    1127
```

Figure 10-15
*Output from
`onstat -g smb c`*

Loading Data Into a Table

You can load data into an existing table in the following ways.

Method to Load Data	TEXT or BYTE Data CLOB or BLOB Data	Reference
DB-Access LOAD statement	Yes	LOAD statement in the <i>IBM Informix Guide to SQL: Syntax</i>
dbload utility	Yes	<i>IBM Informix Migration Guide</i>
dbimport utility	Yes	<i>IBM Informix Migration Guide</i>
ESQL/C programs	Yes	<i>IBM Informix ESQL/C Programmer's Manual</i>
onload utility	No	<i>IBM Informix Migration Guide</i>
onpladm utility	Yes, deluxe mode	IBM Informix Server Administrator
High-Performance Loader (HPL)	Yes, deluxe mode	<i>IBM Informix High-Performance Loader User's Guide</i>



Important: The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been loaded into a database.

Logging and Log Administration

- Chapter 11** **Logging**
- Chapter 12** **Managing Database-Logging Mode**
- Chapter 13** **Logical Log**
- Chapter 14** **Managing Logical-Log Files**
- Chapter 15** **Physical Logging, Checkpoints, and Fast Recovery**
- Chapter 16** **Managing the Physical Log**



Logging

In This Chapter	11-3
Database Server Processes That Require Logging	11-3
Transaction Logging	11-6
Logging of SQL Statements and Database Server Activity	11-6
Activity That is Always Logged	11-7
Activity Logged for Databases with Transaction Logging	11-9
Activity That is Not Logged	11-10
Database-Logging Status	11-10
Unbuffered Transaction Logging.	11-11
Buffered Transaction Logging.	11-12
ANSI-Compliant Transaction Logging.	11-12
No Database Logging	11-12
Databases with Different Log-Buffering Status	11-13
Database Logging in an X/Open DTP Environment	11-13
Settings or Changes for Logging Status or Mode.	11-14

In This Chapter

This chapter describes logging of Informix databases and addresses the following questions:

- Which database server processes require logging?
- What is transaction logging?
- What database server activity is logged?
- What is the database-logging status?
- Who can set or change the database logging status?

All the databases managed by a single database server instance store their log records in the same logical log, regardless of whether they use transaction logging. Most database users might be concerned with whether transaction logging is buffered or whether a table uses logging.

If you want to change the database-logging status, see [“Settings or Changes for Logging Status or Mode” on page 11-14](#).

Database Server Processes That Require Logging

As the Informix database server operates—as it processes transactions, keeps track of data storage, ensures data consistency, and so on—it automatically generates *logical-log records* for some of the actions that it takes. Most of the time the database server makes no further use of the logical-log records. However, when the database server needs to roll back a transaction, to execute a fast recovery after a system failure, for example, the logical-log records are critical. The logical-log records are at the heart of the data-recovery mechanisms.

The database server stores the logical-log records in a *logical log*. The logical log is made up of *logical-log files* that the database server manages on disk until they have been safely transferred offline (*backed up*). The database server administrator keeps the backed up logical-log files until they are needed during a data restore, or until the administrator decides that the records are no longer needed for a restore. For more information, see [Chapter 13, “Logical Log.”](#)

The logical-log records themselves are variable length. This arrangement increases the number of logical-log records that can be written to a page in the logical-log buffer. However, the database server often flushes the logical-log buffer before the page is full. For more information on the format of logical-log records, see the chapter on interpreting logical-log records in the *Administrator's Reference*.

The database server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

- Transaction rollback

If a database is using transaction logging and a transaction must be rolled back, the database server uses the logical-log records to reverse the changes made during the transaction. For more information, see [“Transaction Logging” on page 11-6](#).

- Fast recovery

If the database server shuts down in an uncontrolled manner, the database server uses the logical-log records to recover all transactions that occurred since the oldest update not yet flushed to disk and to roll back any uncommitted transactions. (When all the data in shared memory and on disk are the same, they are *physically consistent*.) The database server uses the logical-log records in fast recovery when it returns the entire database server to a state of logical consistency up to the point of the most recent logical-log record. (For more information, see [“Details of Fast Recovery After A Full Checkpoint” on page 15-20](#).)

- Data restoration

The database server uses the most recent storage-space and logical-log backups to re-create the database server system up to the point of the most recently backed-up logical-log record. The logical restore applies all the log records since the last storage-space backup.

- Deferred checking

If a transaction uses the SET CONSTRAINTS statement to set checking to DEFERRED, the database server does not check the constraints until the transaction is committed. If a constraint error occurs while the transaction is being committed, the database server uses logical-log records to roll back the transaction. For more information, see SET Database Object Mode in the *IBM Informix Guide to SQL: Syntax*.

- Cascading deletes

Cascading deletes on referential constraints use logical-log records to ensure that a transaction can be rolled back if a parent row is deleted and the system fails before the children rows are deleted. For information on table inheritance, see the *IBM Informix Database Design and Implementation Guide*. For information on primary key and foreign key constraints, see the *IBM Informix Guide to SQL: Tutorial*.

- Distributed transactions

Each database server involved in a distributed transaction keeps logical-log records of the transaction. This process ensures data integrity and consistency, even if a failure occurs on one of the database servers that is performing the transaction. For more information, see [“Two-Phase Commit and Logical-Log Records” on page 22-23](#).

- High-Availability Data Replication (HDR)

High-Availability Data Replication uses logical-log records to maintain consistent data on two different database servers so that one of the database servers can be used quickly as a backup database server if the other fails. For more details, see [“How HDR Works” on page 19-9](#).

- Enterprise Replication

You must use database logging with Enterprise Replication because it replicates the data from the logical-log records. For more information, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

Transaction Logging

A database or table is said to *have* or *use* transaction logging when SQL data manipulation statements in a database generate logical-log records.

The database-logging *status* indicates whether a database uses transaction logging. The *log-buffering mode* indicates whether a database uses buffered or unbuffered logging, or ANSI-compliant logging. For more information, see [“Database-Logging Status” on page 11-10](#) and [Chapter 12, “Managing Database-Logging Mode.”](#)

When you create a database, you specify whether it uses *transaction logging* and, if it does, what log-buffering mechanism it uses. After the database is created, you can turn off database logging or change to buffered logging, for example. Even if you turn off transaction logging for all databases, the database server always logs some events. For more information, see [“Activity That is Always Logged” on page 11-7](#) and [“Database Logging in an X/Open DTP Environment” on page 11-13](#).

You can use logging or nonlogging tables within a database. The user who creates the table specifies the type of table. Even if you use nonlogging tables, the database server always logs some events. For more information, see [“Table Types for Dynamic Server” on page 9-38](#).

Logging of SQL Statements and Database Server Activity

Three types of logged activity are possible in the database server:

- Activity that is always logged
- Activity that is logged only for databases that use transaction logging
- Activity that is never logged

Activity That is Always Logged

Some database operations always generate logical-log records, even if you turn off transaction logging or use nonlogging tables.

The following operations are always logged for permanent tables:

- SQL data definition statements:
- Storage-space backups
- Checkpoints
- Administrative changes to the database server configuration such as adding a chunk or dbspace
- Allocation of new extents to tables
- A change to the logging status of a database
- Smart-large-object operations:
 - Creating
 - Deleting
 - Allocating and deallocating extents
 - Truncating
 - Combining and splitting chunk free list pages
 - Changing the LO header and the LO reference count
- Sbspace metadata
- Blobspaces

For SQL data-definition statements, see [Figure 11-1](#).

Figure 11-1
SQL Data-Definition Statements

ALTER ACCESS METHOD	CREATE ROUTINE	DROP TABLE
ALTER FRAGMENT	CREATE ROW TYPE	DROP TRIGGER
ALTER FUNCTION	CREATE SCHEMA	DROP TYPE
ALTER INDEX	CREATE SYNONYM	DROP VIEW

(1 of 2)

Activity That is Always Logged

ALTER OPAQUE TYPE	CREATE TABLE	GRANT
ALTER PROCEDURE	CREATE TEMPORARY TABLE	GRANT FRAGMENT
ALTER ROUTINE	CREATE TRIGGER	RENAME COLUMN
ALTER TABLE	CREATE VIEW	RENAME DATABASE
CREATE ACCESS METHOD	DROP ACCESS METHOD	RENAME INDEX
CREATE AGGREGATE	DROP AGGREGATE	RENAME TABLE
CREATE CAST	DROP CAST	REVOKE
CREATE DATABASE	DROP DATABASE	REVOKE FRAGMENT
CREATE DISTINCT TYPE	DROP FUNCTION	ROLLBACK WORK
CREATE EXTERNAL TABLE	DROP INDEX	START VIOLATIONS TABLE
CREATE FUNCTION	DROP OPCLASS	TRUNCATE TABLE
CREATE INDEX	DROP PROCEDURE	STOP VIOLATIONS TABLE
CREATE OPAQUE TYPE	DROP ROLE	UPDATE STATISTICS
CREATE OPCLASS	DROP ROUTINE	
CREATE PROCEDURE	DROP ROW TYPE	
CREATE ROLE	DROP SYNONYM	

(2 of 2)

Activity Logged for Databases with Transaction Logging

If a database uses transaction logging, the following SQL statements generate one or more log records. If these statements are rolled back, the rollback also generates log records.

DELETE	LOAD	UNLOAD
FLUSH	PUT	UPDATE
INSERT	SELECT INTO TEMP	

The following SQL statements generate log records in special situations.

BEGIN WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
COMMIT WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
EXECUTE	Whether this statement generates a log record depends on the command being executed.
EXECUTE FUNCTION	Whether this statement generates a log record depends on the function being executed.
EXECUTE IMMEDIATE	Whether this statement generates a log record depends on the command being executed.
EXECUTE PROCEDURE	Whether this statement generates a log record depends on the procedure being executed.

Activity That is Not Logged

The following SQL statements do not produce log records, regardless of the database logging mode.

ALLOCATE COLLECTION	DEALLOCATE ROW	LOCK TABLE
ALLOCATE DESCRIPTOR	DECLARE	OPEN
ALLOCATE ROW	DESCRIBE	OUTPUT
CLOSE	DISCONNECT	PREPARE
CONNECT	FETCH	SELECT
DATABASE	FREE	SET ...
DEALLOCATE COLLECTION	GET DESCRIPTOR	UNLOCK TABLE
DEALLOCATE DESCRIPTOR	GET DIAGNOSTICS	WHENEVER
DEALLOCATE ROW	INFO	

For temp tables in temporary dbspaces, nothing is logged, not even the SQL statements listed in [“Activity That is Always Logged” on page 11-7](#). If you include temporary (nonlogging) dbspaces in DBSPACETEMP, the database server places nonlogging tables in these temporary dbspaces first. For more information, see [“Temporary Tables” on page 9-51](#).

Database-Logging Status

You must use transaction logging with a database to take advantage of any of the features listed in [“Database Server Processes That Require Logging” on page 11-3](#).

Every database that the database server manages has a logging status. The logging status indicates whether the database uses transaction logging and, if so, which log-buffering mechanism the database employs. To find out the transaction-logging status of a database, use the database server utilities, as explained in [“Monitoring the Logging Mode of a Database” on page 12-11](#). The database-logging status indicates any of the following types of logging:

- Unbuffered transaction logging
- Buffered transaction logging
- ANSI-compliant transaction logging
- No logging

All logical-log records pass through the logical-log buffer in shared memory before the database server writes them to the logical log on disk. However, the point at which the database server flushes the logical-log buffer is different for buffered transaction logging and unbuffered transaction logging. For more information, see [“How the Database Server Uses Shared Memory” on page 7-6](#) and [“Flushing the Logical-Log Buffer” on page 7-45](#).

Unbuffered Transaction Logging

If transactions are made against a database that uses unbuffered logging, the records in the logical-log buffer are guaranteed to be written to disk during commit processing. When control returns to the application after the COMMIT statement (and before the PREPARE statement for distributed transactions), the logical-log records are on the disk. The database server flushes the records as soon as any transaction in the buffer is committed (that is, a commit record is written to the logical-log buffer).

When the database server flushes the buffer, only the used pages are written to disk. Used pages include pages that are only partially full, however, so some space is wasted. For this reason, the logical-log files on disk fill up faster than if all the databases on the same database server use buffered logging.

Unbuffered logging is the best choice for most databases because it guarantees that all committed transactions can be recovered. In the event of a failure, only uncommitted transactions at the time of the failure are lost. However, with unbuffered logging, the database server flushes the logical-log buffer to disk more frequently, and the buffer contains many more partially full pages, so it fills the logical log faster than buffered logging does.

Buffered Transaction Logging

If transactions are made against a database that uses buffered logging, the records are held (*buffered*) in the logical-log buffer for as long as possible. They are not flushed from the logical-log buffer in shared memory to the logical log on disk until one of the following situations occurs:

- The buffer is full.
- A commit on a database with unbuffered logging flushes the buffer.
- A checkpoint occurs.
- The connection is closed.

If you use buffered logging and a failure occurs, you cannot expect the database server to recover the transactions that were in the logical-log buffer when the failure occurred. Thus, you could lose some committed transactions. In return for this risk, performance during alterations improves slightly. Buffered logging is best for databases that are updated frequently (when the speed of updating is important), as long as you can re-create the updates in the event of failure. You can tune the size of the logical-log buffer to find an acceptable balance for your system between performance and the risk of losing transactions to system failure.

ANSI-Compliant Transaction Logging

The ANSI-compliant database logging status indicates that the database owner created this database using the MODE ANSI keywords. ANSI-compliant databases always use unbuffered transaction logging, enforcing the ANSI rules for transaction processing. You cannot change the buffering status of ANSI-compliant databases.

No Database Logging

If you turn off logging for a database, transactions are not logged, but other operations are logged. For more information, see [“Activity That is Always Logged” on page 11-7](#). Usually, you would turn off logging for a database when you are loading data, or just executing queries.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of database server processing. For example, if you are loading many rows into a database from a recoverable source such as tape or an ASCII file, you might not need transaction logging, and the loading would proceed faster without it. However, if other users are active in the database, you would not have logical-log records of their transactions until you reinitiate logging, which must wait for a level-0 backup.

Databases with Different Log-Buffering Status

All databases on a database server use the same logical log and the same logical-log buffers. Therefore, transactions against databases with different log-buffering statuses can write to the same logical-log buffer. In that case, if transactions exist against databases with buffered logging *and* against databases with unbuffered logging, the database server flushes the buffer *either* when it is full *or* when transactions against the databases with unbuffered logging complete.

Database Logging in an X/Open DTP Environment

Databases in the X/Open distributed transaction processing (DTP) environment must use *unbuffered logging*. Unbuffered logging ensures that the database server logical logs are always in a consistent state and can be synchronized with the transaction manager. If a database created with buffered logging is opened in an X/Open DTP environment, the database status automatically changes to unbuffered logging. The database server supports both ANSI-compliant and non-ANSI databases. For more information, see [“Transaction Managers” on page 22-3](#).

Settings or Changes for Logging Status or Mode

The user who creates a database with the CREATE DATABASE statement establishes the logging status or buffering mode for that database. For more information on the CREATE DATABASE statement, see the *IBM Informix Guide to SQL: Syntax*.

If the CREATE DATABASE statement does not specify a logging status, the database is created without logging.

Only the database server administrator can change logging status. [Chapter 12, “Managing Database-Logging Mode,”](#) describes this topic. Ordinary end users cannot change database-logging status.

If a database does not use logging, you do not need to consider whether buffered or unbuffered logging is more appropriate. If you specify logging but do not specify the buffering mode for a database, the default is unbuffered logging.

End users *can* switch from unbuffered to buffered (but not ANSI-compliant) logging and from buffered to unbuffered logging for the *duration of a session*. The SET LOG statement performs this change within an application. For more information on the SET LOG statement, see the *IBM Informix Guide to SQL: Syntax*.

Managing Database-Logging Mode

In This Chapter	12-3
Changing Database-Logging Mode	12-4
Modifying Database-Logging Mode with ondblog	12-5
Changing Buffering Mode with ondblog	12-5
Canceling a Logging Mode Change with ondblog	12-6
Ending Logging with ondblog	12-6
Making a Database ANSI Compliant with ondblog	12-6
Changing the Logging Mode of an ANSI-Compliant Database	12-7
Modifying Database Logging Mode with ontape	12-7
Turning On Transaction Logging with ontape	12-7
Ending Logging with ontape	12-8
Changing Buffering Mode with ontape	12-8
Making a Database ANSI Compliant with ontape	12-8
Modifying Database Logging Mode with ISA.	12-9
Modifying Database Logging Mode with ON-Monitor	12-9
Modifying the Table-Logging Mode	12-10
Altering a Table to Turn Off Logging	12-10
Altering a Table to Turn On Logging	12-10
Monitoring Transactions	12-11
Monitoring the Logging Mode of a Database	12-11
Monitoring the Logging Mode with SMI Tables	12-11
Monitoring the Logging Mode with ON-Monitor	12-12
Monitoring the Logging Mode with ISA	12-12

In This Chapter

This chapter covers the following topics on changing the database-logging mode:

- Understanding database-logging mode
- Modifying database-logging mode with **ondblog**
- Modifying database-logging mode with **ontape**
- Modifying database-logging mode with ON-Monitor
- Monitoring transaction logging

As a database server administrator, you can alter the logging mode of a database as follows:

- Change transaction logging from buffered to unbuffered.
- Change transaction logging from unbuffered to buffered.
- Make a database ANSI compliant.
- Add transaction logging (buffered or unbuffered) to a database.
- End transaction logging for a database.

For information about database-logging mode, when to use transaction logging, and when to buffer transaction logging, see [Chapter 11, “Logging.”](#) To find out the current logging mode of a database, see [“Monitoring the Logging Mode of a Database” on page 12-11.](#)

Changing Database-Logging Mode

You can use **ondblog**, **ontape**, or ISA to add or change logging. Then use ON-Bar, or **ontape** to back up the data. When you use ON-Bar or **ontape**, the database server must be in online or quiescent mode.

For information on ON-Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

Figure 12-1 shows how the database server administrator can change the database-logging mode. Certain logging mode changes take place immediately, while other changes require a level-0 backup.

Figure 12-1
Logging Mode Transitions

Converting from:	Converting to:			
	No Logging	Unbuffered Logging	Buffered Logging	ANSI Compliant
No logging	Not applicable	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)
Unbuffered logging	Yes	Not applicable	Yes	Yes
Buffered logging	Yes	Yes	Not applicable	Yes
ANSI compliant	Illegal	Illegal	Illegal	Not applicable

Some general points about changing the database-logging mode follow:

- While the logging status is being changed, the database server places an exclusive lock on the database to prevent other users from accessing the database, and frees the lock when the change is complete.
- If a failure occurs during a logging-mode change, check the logging mode in ISA or the flags in the **sysdatabases** table in the **sysmaster** database, after you restore the database server data. For more information, see [“Monitoring the Logging Mode of a Database” on page 12-11](#). Then retry the logging-mode change.
- Once you choose either buffered or unbuffered logging, an application can use the SQL statement SET LOG to change from one logging mode to the other. This change lasts for the duration of the session. For information on SET LOG, see the *IBM Informix Guide to SQL: Syntax*.
- If you add logging to a database, the change is not complete until the next level-0 backup of all the storage spaces for the database.

Modifying Database-Logging Mode with ondblog

You can use the **ondblog** utility to change the logging mode for one or more databases. If you add logging to a database, you must create a level-0 backup of the dbspace(s) that contains the database before the change takes effect. For more information, see the section on using **ondblog** in the *Administrator's Reference*.

Changing Buffering Mode with ondblog

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, execute the following command:

```
ondblog unbuf stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, execute the following command:

```
ondblog buf stores_demo
```

Canceling a Logging Mode Change with ondblog

To cancel the logging mode change request before the next level-0 backup occurs, execute the following command:

```
ondblog cancel stores_demo
```

Note that you cannot cancel the logging changes that are executed immediately.

Ending Logging with ondblog

To end logging for two databases that are listed in a file called **dbfile**, execute the following command:

```
ondblog nolog -f dbfile
```

Making a Database ANSI Compliant with ondblog

To make a database called **stores_demo** into an ANSI-compliant database with **ondblog**, execute the following command:

```
ondblog ansi stores_demo
```

Changing the Logging Mode of an ANSI-Compliant Database

Once you create or convert a database to ANSI mode, you cannot easily change it to any other logging mode. If you accidentally convert a database to ANSI mode, follow these steps to change the logging mode:

To change the logging mode

1. To unload the data, use **dbexport** or any other migration utility. The **dbexport** utility creates the **schema** file.
For information about how to load and unload data, see the *IBM Informix Migration Guide*.
2. To re-create a database with buffered logging and load the data, use the **dbimport -l buffered** command.
To re-create a database with unbuffered logging and load the data, use the **dbimport -l** command.

Modifying Database Logging Mode with ontape

If you use **ontape** as your backup tool, you can use **ontape** to change the logging mode of a database.

Turning On Transaction Logging with ontape

Before you modify the database-logging mode, read [“Changing Database-Logging Mode” on page 12-4](#).

You add logging to a database with **ontape** at the same time that you create a level-0 backup.

For example, to add buffered logging to a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -s -B stores_demo
```



To add unbuffered logging to a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -s -U stores_demo
```

In addition to turning on transaction logging, these commands create full-system storage-space backups. When **ontape** prompts you for a backup level, specify a level-0 backup.

Tip: With **ontape**, you must perform a level-0 backup of all storage spaces.

Ending Logging with **ontape**

To end logging for a database called **stores_demo** with **ontape**, execute the following command:

```
ontape -N stores_demo
```

Changing Buffering Mode with **ontape**

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, execute the following command:

```
ontape -U stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, execute the following command:

```
ontape -B stores_demo
```

Making a Database ANSI Compliant with **ontape**

To make a database called **stores_demo**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database with **ontape**, execute the following command:

```
ontape -A stores_demo
```

To make a database called **stores_demo**, which does not already use transaction logging, into an ANSI-compliant database with **ontape**, execute the following command:

```
ontape -s -A stores_demo
```

In addition to making a database ANSI compliant, this command also creates a storage-space backup at the same time. Specify a level-0 backup when you are prompted for a level.



Tip: Once you change the logging mode to ANSI compliant, you cannot easily change it again. To change the logging mode of ANSI-compliant databases, unload the data, re-create the database with the new logging mode, and reload the data. For details, see [“Changing the Logging Mode of an ANSI-Compliant Database” on page 12-7.](#)

Modifying Database Logging Mode with ISA

ISA uses the **ondblog** utility to modify the database-logging mode. If you turn on logging, perform a level-0 backup. For more information, see the ISA online help and [“Modifying Database-Logging Mode with ondblog” on page 12-5.](#)

UNIX

Modifying Database Logging Mode with ON-Monitor

You can use ON-Monitor to change the log-buffering mode between unbuffered and buffered. If you want to add logging to a database or make it ANSI compliant, you cannot use ON-Monitor. You must use **ontape**.

To change the log-buffering mode for a database, select the **Logical-Logs→Databases** option.

Modifying the Table-Logging Mode

The database server creates standard tables that use logging by default. To create a nonlogging table, use the CREATE TABLE statement with the WITH LOG clause. For information on the CREATE TABLE and ALTER TABLE Statements, see the *IBM Informix Guide to SQL: Syntax*. For more information, refer to [“Table Types for Dynamic Server” on page 9-38](#).

Altering a Table to Turn Off Logging

To switch a table from logging to nonlogging, use the SQL statement ALTER TABLE with the TYPE option of RAW. For example, the following statement changes table **tablog** to a RAW table:

```
ALTER TABLE tablog TYPE (RAW)
```

Altering a Table to Turn On Logging

To switch from a nonlogging table to a logging table, use the SQL statement ALTER TABLE with the TYPE option of STANDARD. For example, the following statement changes table **tabnolog** to a STANDARD table:

```
ALTER TABLE tabnolog TYPE (STANDARD)
```



Warning: When you alter a table to STANDARD, you turn logging on for that table. After you alter the table, perform a level-0 backup if you need to be able to restore the table.

Monitoring Transactions

This section discusses ways to monitor transactions.

Command	Description	Reference
<code>onstat -x</code>	Monitor transactions.	“Monitoring a Global Transaction” on page 22-22
<code>onstat -g sql</code>	Monitor SQL statements, listed by session ID and database.	Performance monitoring in the <i>Performance Guide</i>
<code>onstat -g stm</code>	Monitor memory usage of prepared SQL statements.	Memory utilization in the <i>Performance Guide</i>

Monitoring the Logging Mode of a Database

This section discusses ways to monitor the logging mode of your database and tables.

Monitoring the Logging Mode with SMI Tables

Query the `sysdatabases` table in the `sysmaster` database to determine the logging mode. This table contains a row for each database that the database server manages. The `flags` field indicates the logging mode of the database. The `is_logging`, `is_buff_log`, and `is_ansi` fields indicate whether logging is active, and whether buffered logging or ANSI-compliant logging is used. For a description of the columns in this table, see the `sysdatabases` section in the chapter about the `sysmaster` database in the *Administrator’s Reference*.

UNIX

Monitoring the Logging Mode with ON-Monitor

To use ON-Monitor to find out the logging mode of a database, select the **Status→Databases** option. When database logging is on, ON-Monitor shows the buffering mode. ON-Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in [“Monitoring the Logging Mode with SMI Tables” on page 12-11](#).

Monitoring the Logging Mode with ISA

To use ISA to display the logging status and buffering mode for your databases, select **SQL→Schema**.

Logical Log

In This Chapter	13-3
What Is the Logical Log?	13-3
Location of Logical-Log Files	13-4
Identification of Logical-Log Files.	13-5
Status Flags of Logical-Log Files	13-6
Size of the Logical Log	13-7
Number of Logical-Log Files	13-7
Performance Considerations	13-8
Dynamic Log Allocation	13-9
Freeing of Logical-Log Files	13-10
Action If the Next Logical-Log File Is Not Free	13-10
Action if the Next Log File Contains the Last Checkpoint	13-11
Logging Blobspaces and Simple Large Objects	13-11
Switching Log Files to Activate Blobspaces	13-12
Backing Up Log Files to Free Blobpages	13-12
Backing Up Blobspaces After Inserting or Deleting TEXT and BYTE Data	13-13
Logging Sbspaces and Smart Large Objects	13-13
Using Sbspace Logging	13-13
Choosing Logging for Smart Large Objects	13-14
Logging for Updated Smart Large Objects	13-14
Turning Logging On or Off for an Sbspace	13-15

Using Smart-Large-Object Log Records	13-16
Preventing Long Transactions When Logging Smart-Large-Object Data	13-16
Logging Process	13-17
Dbospace Logging	13-17
Blobspace Logging.	13-17

In This Chapter

The information in [Chapter 11, “Logging,”](#) and this chapter will help you understand how the database server uses the logical log. For information on how to perform logical-log tasks, see [Chapter 14, “Managing Logical-Log Files,”](#) and [Chapter 12, “Managing Database-Logging Mode.”](#)

What Is the Logical Log?

To keep a history of transactions and database server changes since the time of the last storage-space backup, the database server generates log records. The database server stores the log records in the *logical log*, a circular file that is composed of three or more logical-log files. The log is called *logical* because the log records represent logical operations of the database server, as opposed to physical operations. At any time, the combination of a storage-space backup plus logical-log backup contains a complete copy of your database server data.

As the database server administrator, you must configure and manage the logical log. For example, if you do not back up the log files regularly, the logical log fills and the database server suspends processing.

These responsibilities include the following tasks:

- Choosing an appropriate location for the logical log
See [“Location of Logical-Log Files”](#) on page 13-4.
- Monitoring the logical-log file status
See [“Identification of Logical-Log Files”](#) on page 13-5.
- Allocating an appropriate amount of disk space for the logical log
See [“Size of the Logical Log”](#) on page 13-7.

- Allocating additional log files whenever necessary
See [“Allocating Log Files” on page 14-14.](#)
- Backing up the logical-log files to media
See [“Backing Up Logical-Log Files” on page 14-6](#) and [“Freeing of Logical-Log Files” on page 13-10.](#)
- Managing logging of blobspaces and sbspaces
See [“Logging Blobspaces and Simple Large Objects” on page 13-11](#) and [“Logging Sbspaces and Smart Large Objects” on page 13-13.](#)

Location of Logical-Log Files

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no control over this action. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize contention), move the logical-log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log. See [“Moving a Logical-Log File to Another Dbspace” on page 14-20.](#)

To improve performance further, separate the logical-log files into two groups and store them on two separate disks (neither of which contains data). For example, if you have six logical-log files, you might locate files 1, 3, and 5 on disk 1, and files 2, 4, and 6 on disk 2. This arrangement improves performance because the same disk drive never has to handle writes to the current logical-log file and backups to tape at the same time.

The logical-log files contain critical information and should be mirrored for maximum data protection. If you move logical-log files to a different dbspace, plan to start mirroring on that dbspace.

Identification of Logical-Log Files

Each logical-log file, whether backed up to media or not, has a *unique ID* number. The sequence begins with 1 for the first logical-log file filled after you initialize the database server disk space. When the current logical-log file becomes full, the database server switches to the next logical-log file and increments the unique ID number for the new log file by one. Log files that are newly added or marked for deletion have unique ID numbers of 0.

The actual disk space allocated for each logical-log file has an identification number known as the *log file number*. For example, if you configure six logical-log files, these files have log numbers one through six. The log numbers might be out of sequence. As logical-log files are backed up and freed, the database server reuses the disk space for the logical-log files.

Figure 13-1 illustrates the relationship between the log numbers and the unique ID numbers. Log 7 is inserted after log 5 and used for the first time in the second rotation.

Figure 13-1
Logical-Log File-Numbering Sequence

Log File Number	First Rotation Unique ID Number	Second Rotation Unique ID Number	Third Rotation Unique ID Number
1	1	7	14
2	2	8	15
3	3	9	16
4	4	10	17
5	5	11	18
7	0	12	19
6	6	13	20

Status Flags of Logical-Log Files

All logical-log files have one of the following status flags in the first position: Added (A), Deleted (D), Free (F), or Used (U). [Figure 13-2](#) shows the possible log-status flag combinations.

Figure 13-2
Logical-Log Status Flags

Status Flag	Status of Logical-Log File
A-----	Log file has been <i>added</i> , and is available, but has not yet been used.
D-----	If you drop a log file with a status of U-B, it is marked as <i>deleted</i> . This log file is dropped and its space is freed for reuse when you take a level-0 backup of all storage spaces.
F-----	Log file is <i>free</i> and available for use. A logical-log file is freed after it is backed up, all transactions within the logical-log file are closed, and the oldest update stored in this file is flushed to disk.
U	Log file has been <i>used</i> but not backed up.
U-B----	Log file is <i>backed up</i> but still needed for recovery. (The log file is freed when it is no longer needed for recovery.)
U-B---L	Log is backed up but still needed for recovery. Contains the last checkpoint record.
U---C	The database server is <i>currently</i> filling the log file.
U---C-L	This current log file contains the <i>last</i> checkpoint record.

Use the **onstat -l** command to list the log files by number and monitor the status flags and percentage of log space used. For more details, see [“onstat -l” on page 14-11](#).

Size of the Logical Log

Decide how many log files you want and what size. If you allocate more disk space than necessary, space is wasted. If you do not allocate enough disk space, however, performance might be adversely affected. Consider these points about the size and number of the logical-log files:

- The minimum size for a logical-log file is 200 kilobytes.
- The maximum size for a logical-log file is 1048576 pages (equivalent to 0x100000).
- Smaller log files mean that you can recover to a later time if the disk that contains the log files goes down. If continuous log backup is set, log files are automatically backed up as they fill. Smaller logs result in slightly longer logical recovery.
- Use larger log files when many users are writing to the logs at the same time.

Number of Logical-Log Files

When you think about the number of logical-log files, consider these points:

- You must always have at least three logical-log files and a maximum of 32,767 log files. The number of log files depends on the size of the log files.
- The number of log files affects the frequency of logical-log backups.
- The number of logical-log files affects the rate at which blob space blobpages can be reclaimed. See [“Backing Up Log Files to Free Blobpages” on page 13-12](#).

Performance Considerations

For a given level of system activity, the less logical-log disk space that you allocate, the sooner that logical-log space fills up, and the greater the likelihood that user activity is blocked due to backups and checkpoints. Tune the logical-log size to find the optimum value for your system.

- Logical-log backups

When the logical-log files fill, you have to back them up. The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. Put the physical log, logical logs, and user data on separate disks. (See the *IBM Informix Backup and Restore Guide*.)

- Checkpoints

Checkpoints block user processing briefly. If log files are backed up and freed more frequently, the checkpoints occur more frequently.

- Size of the logical log

A smaller logical log fills faster than a larger logical log. You can add a larger logical-log file, as explained in [“Adding Logical-Log Files Manually” on page 14-16](#).

- Size of individual logical-log records

The sizes of the logical-log records vary, depending on both the processing operation and the database server environment. In general, the longer the data rows, the larger the logical-log records. The logical log contains images of rows that have been inserted, updated, or deleted. Updates can use up to twice as much space as inserts and deletes because they might contain both before-images and after-images. (Inserts store only the after-image and deletes store only the before-image.)

- Number of logical-log records

The more logical-log records written to the logical log, the faster it fills. Databases with transaction logging fill the logical log faster than transactions against databases without transaction logging.

- Type of log buffering

Databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.

- Enterprise Replication on a table

Because Enterprise Replication generates before-images and after-images of the replicated tables, it could cause the logical log to fill.

- Frequency of rollbacks

More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small.

- Number of smart large objects

Smart large objects that have user data logging enabled and have a large volume of user data updates can fill logical logs at a prodigious rate. Use temporary smart large objects if you do not want to log the metadata.

Dynamic Log Allocation

Dynamic log allocation prevents log files from filling and hanging the system during long transaction rollbacks. The only time that this feature becomes active is when the next log file contains an open transaction. (A *transaction* is *long* if it is not committed or rolled back when it reaches the long-transaction high-watermark.)

The database server automatically (dynamically) allocates a log file after the current log file when the next log file contains an open transaction. Dynamic log allocation allows you to do the following actions:

- Add a log file while the system is active
- Insert a log file after the current log file
- Immediately access new log files even if the root dbspace is not backed up

The best way to test dynamic log allocation is to produce a transaction that spans all the log files and then use **onstat -l** to check for newly added log files. For more information, see [“Allocating Log Files” on page 14-14](#).

Important: *You still must back up log files to prevent them from filling. If the log files fill, the system hangs until you perform a backup.*



Freeing of Logical-Log Files

Each time the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began. The following criteria must be satisfied before the database server frees a logical-log file for reuse:

- The log file is backed up.
- No records within the logical-log file are associated with open transactions.
- The logical-log file does not contain the oldest update not yet flushed to disk.

Action If the Next Logical-Log File Is Not Free

If the database server attempts to switch to the next logical-log file but finds that the next log file in sequence is still in use, the database server immediately suspends all processing. Even if other logical-log files are free, the database server cannot skip a file in use and write to a free file out of sequence. Processing stops to protect the data within the logical-log file.

The logical-log file might be in use for any of the following reasons:

- The file contains the latest checkpoint or the oldest update not yet flushed to disk.
Issue the **onmode -c** command to perform a full checkpoint and free the logical-log file. For more information, see [“Forcing a Full Checkpoint” on page 16-9](#).
- The file contains an open transaction.
The open transaction is the long transaction discussed in [“Setting High-Watermarks for Rolling Back Long Transactions” on page 14-25](#).
- The file is not backed up.
If the logical-log file is not backed up, processing resumes when you use ON-Bar or **ontape** to back up the logical-log files.

Action if the Next Log File Contains the Last Checkpoint

The database server does not suspend processing when the next log file contains the last checkpoint or the oldest update. The database server always forces a full checkpoint when it switches to the last available log, if the previous checkpoint record or oldest updated not-yet-flushed-to-disk is located in the log that follows the last available log. For example, if four logical-log files have the status shown in the following list, the database server forces a checkpoint when it switches to logical-log file 3.

Log File Number	Logical-Log File Status
1	U-B----
2	U---C--
3	F
4	U-B---L

Logging Blobspaces and Simple Large Objects

Simple-large-object data (TEXT and BYTE data types) is potentially too voluminous to include in a logical-log record. If simple large objects are always logged, they might be so large that they slow down the logical log.

The database server assumes that you designed your databases so that smaller simple large objects are stored in dbspaces and larger simple large objects are stored in blobspaces:

- The database server includes simple-large-object data in log records for simple large objects stored in dbspaces.
- The database server does not include simple-large-object data in log records for simple large objects stored in blobspaces. The logical log records blobspace data only when you back up the logical logs.

To obtain better overall performance for applications that perform frequent updates of simple large objects in blobspaces, reduce the size of the logical log. Smaller logs can improve access to simple large objects that must be reused. For more information, see the chapter on configuration effects on I/O utilization in your *Performance Guide*.

Switching Log Files to Activate Blobspaces

You must switch to the next logical-log file in these situations:

- After you create a blobspace, if you intend to insert simple large objects in the blobspace right away
- After you add a new chunk to an existing blobspace, if you intend to insert simple large objects in the blobspace that will use the new chunk

The database server requires that the statement that creates a blobspace, the statement that creates a chunk in the blobspace, and the statements that insert simple large objects into that blobspace appear in separate logical-log files. This requirement is independent of the database-logging status.

For instructions on switching to the next log file, see [“Switching to the Next Logical-Log File” on page 14-7](#).

Backing Up Log Files to Free Blobpages

When you delete data stored in blobspace pages, those pages are not necessarily freed for reuse. The blobspace pages are free only when *both* of the following actions have occurred:

- The TEXT or BYTE data has been deleted, either through an UPDATE to the column or by deleting the row
- The logical log that stores the INSERT of the row that has TEXT or BYTE data is backed up

Backing Up Blobspaces After Inserting or Deleting TEXT and BYTE Data

Be sure to back up all blobspaces and logical logs containing transactions on simple large objects stored in a blobspace. During log backup, the database server uses the data pointer in the logical log to copy the changed TEXT and BYTE data from the blobspace into the logical log.

Logging Sbspaces and Smart Large Objects

Sbspaces, described in [“Sbspaces” on page 9-21](#), contain two components: metadata and user data. By default, sbspaces are *not* logged.

The *metadata* component of the sbspace describes critical characteristics of smart large objects stored in a particular sbspace. The metadata contains pointers to the smart large objects. If the metadata were to be damaged or become inaccessible, the sbspace would be corrupted and the smart large objects within that sbspace would be unrecoverable.

Metadata in a standard sbspace is *always* logged, even if logging is turned off for a database. Logging sbspace metadata ensures that the metadata can always be recovered to a consistent transaction state. However, metadata in a temporary sbspace is *not* logged.

Using Sbspace Logging

When an sbspace is logged, the database server slows down, and the logical logs fill up quickly. If you use logging for sbspaces, you must ensure that the logical logs are large enough to hold the logging data. For more information, see [“Estimating the Log Size When Logging Smart Large Objects” on page 14-5](#).

When you turn on logging for a database, the database server does not begin logging until you perform a level-0 backup. However, when you turn on logging for a smart large object, the database server begins logging changes to it immediately. To reduce the volume of log entries, you could load smart large objects with logging turned off and then turn logging back on to capture updates to the smart large objects.



Warning: When you turn logging on for a smart large object, you must immediately perform a level-0 backup to be able to recover and restore the smart large object.

For more information, see [“Backing Up Sbspaces” on page 14-7](#) and the *IBM Informix Backup and Restore Guide*.

Choosing Logging for Smart Large Objects

Use logging for smart large objects if users are updating the data frequently or if the ability to recover any updated data is critical. The database server writes a record of the operation (insert, update, delete, read, or write) to the logical-log buffer. The modified portion of the CLOB or BLOB data is included in the log record.

To increase performance, turn off logging for smart large objects. Also turn off logging if users are primarily analyzing the data and updating it infrequently, or if the data is not critical to recover.

Logging for Updated Smart Large Objects

When you update a smart large object, the database server does not log the entire object. Assume that the user is writing X bytes of data at offset Y with logging enabled for smart large objects. The database server logs the following:

- If Y is set to the end of the large object, the database server logs X bytes (the updated byte range).
- If Y is at the beginning or in the middle of the large object, the database server logs the smallest of these choices:
 - Difference between the old and new image
 - Before-image and after-image
 - Nothing is logged if the before- and after-images are the same

Turning Logging On or Off for an Sbspace

If you want to use logging in an sbspace, specify the **-Df LOGGING=ON** option of the **onspaces** command when you create the sbspace. If logging is turned off in the sbspace, you can turn on logging for smart large objects in specific columns. One column that contains smart large objects could have logging turned on while another column has logging turned off.

To verify that smart large objects in an sbspace are logged, use the following command:

```
oncheck -pS sbspace | grep "Create Flags"
```

If you create smart large objects in the sbspace with the default logging option and you see the LO_NOLOG flag in the output, the smart large objects in this sbspace are not logged. If you see the LO_LOG flag in the output, all smart large objects in this sbspace are logged.

You can modify the logging status of an sbspace in any of the following ways.

Function or Statement to Specify	Logging Action	References
onspaces -ch -Df LOGGING=ON	Turns logging on or off for an existing sbspace	“Altering Storage Characteristics of Smart Large Objects” on page 10-27 <i>Administrator’s Reference</i>
LOG option in the PUT clause of the CREATE TABLE or ALTER TABLE statement	Turns on logging for all smart large objects that you load into the column	“Logging” on page 9-27 <i>IBM Informix Guide to SQL: Syntax</i>
mi_lo_create DataBlade API function	Turns off logging for a smart large object when it is initially loaded	<i>IBM Informix DataBlade API Function Reference</i>
mi_lo_alter DataBlade API function	Turns on logging after the load is complete	<i>IBM Informix DataBlade API Function Reference</i>
ifx_lo_create ESQ/C function	Turns off logging for a smart large object when it is initially loaded	<i>IBM Informix ESQ/C Programmer’s Manual</i>
ifx_lo_alter ESQ/C function	Turns on logging after the load is complete	<i>IBM Informix ESQ/C Programmer’s Manual</i>



Using Smart-Large-Object Log Records

When you create a smart large object with the LOG option, the logical log creates a *smart-blob log record*. Smart-blob log records track changes to user data or metadata. When smart large objects are updated, only the modified portion of the sbpage appears in the log record. User-data log records appear in the logical log only when logging is enabled for the smart large object.

Warning: *Be careful about enabling logging for smart large objects that are updated frequently. This logging overhead might significantly slow down the database server.*

For information on the log records for smart large objects, see the chapter on interpreting logical-log records in the *Administrator's Reference*.

Preventing Long Transactions When Logging Smart-Large-Object Data

You can use smart large objects in situations where the data collection process for a single smart large object lasts for long periods of time. Consider, for example, an application that records many hours of low-quality audio information. Although the amount of data collected might be modest, the recording session might be long, resulting in a long-transaction condition.

Tip: *To prevent long transactions from occurring, periodically commit writes to smart large objects.*



Logging Process

This section describes in detail the logging process for dbspaces, blobspaces, and sbspaces. This information is not required for performing normal database server administration tasks.

Dbpace Logging

The database server uses the following logging process for operations that involve data stored in dbspaces:

1. Reads the data page from disk to the shared-memory page buffer
2. Copies the unchanged page to the physical-log buffer, if needed
3. Writes the new data to the page buffer and creates a logical-log record of the transaction, if needed
4. Flushes the physical-log buffer to the physical log on disk
5. Flushes the logical-log buffer to a logical-log file on disk
6. Flushes the page buffer and writes it back to disk

Blobspace Logging

The database server logs blobspace data, but the data does not pass through either shared memory or the logical-log files on disk. The database server copies data stored in a blobspace directly from disk to tape. Records of modifications to the blobspace overhead pages (the free-map and bitmap pages) are the only blobspace data that reaches the logical log.

Managing Logical-Log Files

In This Chapter	14-3
Estimating the Size and Number of Log Files	14-4
Estimating the Log Size When Logging Smart Large Objects	14-5
Estimating the Number of Logical-Log Files	14-5
Backing Up Logical-Log Files	14-6
Backing Up Blobspaces	14-6
Backing Up Sbspaces	14-7
Switching to the Next Logical-Log File	14-7
Freeing a Logical-Log File	14-8
Deleting a Log File with Status D	14-8
Freeing a Log File with Status U	14-9
Freeing a Log File with Status U-B or F	14-9
Freeing a Log File with Status U-C or U-C-L	14-10
Freeing a Log File with Status U-B-L	14-10
Monitoring Logging Activity	14-10
Monitoring the Logical Log for Fullness	14-11
onstat -l	14-11
oncheck -pr	14-12
Monitoring Temporary Logical Logs	14-13
Using SMI Tables	14-13
Using ON-Monitor	14-14
Monitoring Log-Backup Status	14-14

Allocating Log Files.	14-14
Adding Logs Dynamically	14-14
Size and Number of Dynamically Added Log Files	14-15
Location of Dynamically Added Log Files	14-16
Adding Logical-Log Files Manually	14-16
Dropping Logical-Log Files	14-18
Changing the Size of Logical-Log Files	14-20
Moving a Logical-Log File to Another Dbspace	14-20
Changing Logging Configuration Parameters.	14-21
Using ON-Monitor to Change LOGFILES.	14-22
Displaying Logical-Log Records	14-23
Monitoring Events for Dynamically Added Logs	14-23
Setting High-Watermarks for Rolling Back Long Transactions	14-25
Long-Transaction High-Watermark (LTXHWM)	14-26
Exclusive Access, Long-Transaction High-Watermark (LTXEHWM)	14-26
Adjusting the Size of Log Files to Prevent Long Transactions	14-27
Recovering from a Long Transaction Hang	14-27

In This Chapter

You must manage logical-log files even if none of your databases uses transaction logging. For background information regarding the logical log, refer to [Chapter 13, “Logical Log.”](#)

You must log in as either **informix** or **root** on UNIX to make any of the changes described in this chapter. You must be a member of the **Informix-Admin** group on Windows.

You would perform these tasks when setting up your logical log:

- Before you initialize the database server, use the LOGFILES parameter to specify the number of logical-log files to create.
- After the database server is online, estimate the size and number of logical-log files that your system will need.
See [“Estimating the Size and Number of Log Files” on page 14-4.](#)
- If you do not want to use the default values, change the LOGSIZE and LOGBUFF configuration parameters.
See [“Changing Logging Configuration Parameters” on page 14-21.](#)
- Add the estimated number of logical-log files.
See [“Allocating Log Files” on page 14-14.](#)

You would perform the following tasks routinely:

- Backing up a logical-log file
- Switching to the next logical-log file
- Freeing a logical-log file
- Monitoring logging activity and log-backup status

You would perform these tasks occasionally, if necessary:

- Adding a logical-log file
- Dropping a logical-log file
- Changing the size of a logical-log file
- Moving a logical-log file
- Changing the logical-log configuration parameters
- Monitoring event alarms for logical logs
- Setting high-watermarks for transactions

Estimating the Size and Number of Log Files

Use the LOGSIZE configuration parameter to set the size of the logical-log files. It is difficult to predict how much logical-log space your database server system requires until it is fully in use.

The easiest way to increase the amount of space for the logical log is to add another logical-log file. See [“Adding Logical-Log Files Manually” on page 14-16](#).

The following expression provides the recommended *minimum* total-log-space configuration, in kilobytes:

$$\text{LOGSIZE} = (((\text{connections} * \text{maxrows}) * \text{rowsize}) / 1024) / \text{LOGFILES}$$

LOGSIZE	Specifies the size of each logical-log file in kilobytes.
<i>connections</i>	Specifies the maximum number of connections for all network types that you specify in the sqlhosts file or registry and in the NETTYPE parameter. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, sum the users fields for each NETTYPE, and substitute this total for <i>connections</i> in the preceding formula.
<i>maxrows</i>	Specifies the largest number of rows to be updated in a single transaction.

(1 of 2)

<i>rowsize</i>	Specifies the average size of a table row in bytes. To calculate the <i>rowsize</i> , add up the length (from the syscolumns system catalog table) of the columns in the row.
1024	Converts the LOGSIZE to units of kilobytes.
LOGFILES	Specifies the number of logical-log files.

(2 of 2)

Estimating the Log Size When Logging Smart Large Objects

If you plan to log smart-large-object user data, you must ensure that the log size is *considerably* larger than the amount of data being written. If you store smart large objects in standard sbspaces, the metadata is always logged, even if the smart large objects are not logged. If you store smart large objects in temporary sbspaces, there is no logging at all.

Estimating the Number of Logical-Log Files

The LOGFILES parameter provides the number of logical-log files at system initialization. If all your logical-log files are the same size, you can calculate the total space allocated to the logical-log files as follows:

```
total logical log space = LOGFILES * LOGSIZE
```

If the database server contains log files of different sizes, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you need to add the sizes for each individual log file on disk. Check the **size** field in the **onstat -l** output. For more information, see [“onstat -l” on page 14-11](#).

For information on LOGSIZE, LOGFILES, and NETTYPE, see the chapter on configuration parameters in the *Administrator's Reference*.

Backing Up Logical-Log Files

The logical logs contain a history of the transactions that have been performed. The process of copying a logical-log file to media is referred to as *backing up* a logical-log file. Backing up logical-log files achieves the following two objectives:

- It stores the logical-log records on media so that they can be rolled forward if a data restore is needed.
- It makes logical-log-file space available for new logical-log records.

If you neglect to back up the log files, you can run out of log space.

You can initiate a manual logical-log backup or set up continuous logical-log backups. After you restore the storage spaces, you must restore the logical logs to bring the data to a consistent state. For more information on log backups, see the *IBM Informix Backup and Restore Guide*.

Backing Up Blobspaces

It does not matter whether you back up the logical logs or blobspaces first.

To back up blobspace data

1. Close the current logical log if it contains transactions on simple large objects in a blobspace.
2. Perform a backup of the logical logs and blobspace as soon as possible after updating simple-large-object data.

Warning: *If you do not back up these blobspaces and logical logs, you might not be able to restore the blobspace data. If you wait until a blobspace is down to perform the log backup, the database server cannot access the blobspace to copy the changed data into the logical log.*



Backing Up Sbspaces

When you turn on logging for smart large objects, you must perform a level-0 backup of the sbspace.

Figure 14-1 shows what happens if you turn on logging in an sbspace that is not backed up. The unlogged changes to smart large object **LO1** are lost during the failure, although the logged changes are recoverable. You will not be able to fully restore **LO1**.

During fast recovery, the database server rolls forward all committed transactions for **LO1**. If **LO1** is unlogged, the database server would be unable to roll back uncommitted transactions. Then the **LO1** contents would be incorrect. For more information, refer to “Fast Recovery” on page 15-18.

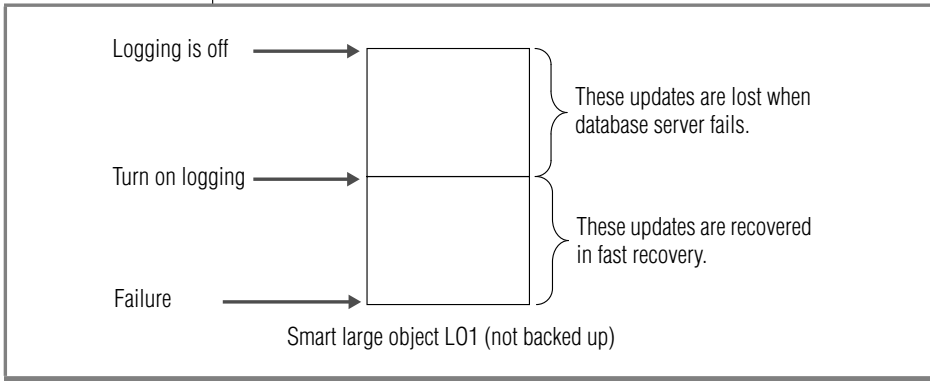


Figure 14-1
*Turning On Logging
in an Sbspace*

Switching to the Next Logical-Log File

You might want to switch to the next logical-log file before the current log file becomes full for the following reasons:

- To back up the current log
- To activate new blobspaces and blobspace chunks

The database server can be in online mode to make this change. Execute the following command to switch to the next available log file:

```
onmode -l
```

The change takes effect immediately. (Be sure that you type a lowercase L on the command line, not a number 1.)

Freeing a Logical-Log File

If a log file is newly added (status **A**), it is immediately available for use. It also can be dropped immediately.

You might want to free a logical-log file for the following reasons:

- So that the database server does not stop processing
- To free the space used by deleted blobpages

The procedures for freeing log files vary, depending on the status of the log file. Each procedure is described in the following sections. To find out the status of logical-log files, see [“Status Flags of Logical-Log Files” on page 13-6](#) and [“Monitoring Logging Activity” on page 14-10](#).



Tip: For information using ON-Bar or *ontape* to back up storage spaces and logical logs, refer to the “IBM Informix Backup and Restore Guide.”

Deleting a Log File with Status D

When you drop a used log file, it is marked as deleted (status **D**) and cannot be used again, and **onparams** prints this message:

```
Log file log_file_number has been pre-dropped. It will be deleted
from the log_list and its space can be reused once you take level
0 archives of all BLOBspaces, Smart BLOBspaces and non-temporary
DBspaces.
```

To delete the log file, create a level-0 backup of all storage spaces.

Freeing a Log File with Status U

If a log file contains records, but is not yet backed up (status **U**), back up the file using the backup tool that you usually use.

If backing up the log file does not change the status to free (**F**), its status changes to either **U-B** or **U-B-L**. See [“Freeing a Log File with Status U-B or F,”](#) following, or [“Freeing a Log File with Status U-B-L”](#) on page 14-10.

Freeing a Log File with Status U-B or F

If a log file is backed up but still in use (status **U-B**), some transactions in the log file are still under way, or the log file contains the oldest update that is required for fast recovery. Because a log file with status **F** has been used in the past, it follows the same rules as for status **U-B**.

To free a backed up log file that is in use

1. If you do not want to wait until the transactions complete, take the database server to quiescent mode. See [“Immediately from Online to Quiescent”](#) on page 4-18. Any active transactions are rolled back.
2. Because a log file with status **U-B** might contain the oldest update, you must use the **onmode -c** command to force a full checkpoint.

A log file that is backed up but *not* in use (status **U-B**) does not need to be freed. In the following example, log 34 does not need to be freed, but logs 35 and 36 do. Log 35 contains the last checkpoint, and log 36 is backed up but still in use.

```
34 U-B-- Log is used, backed up, and not in use
35 U-B-L Log is used, backed up, contains last checkpoint
36 U-B-- Log is used, backed up, and not in use
37 U-C-- This is the current log file, not backed up
```

Tip: You can free a logical log with a status of U-B (and not L) only if it is not spanned by an active transaction and does not contain the oldest update.



Freeing a Log File with Status U-C or U-C-L

Follow these steps to free the current log file.

To free the current log file (status C)

1. Execute the following command to switch the current log file to the next available log file:

```
onmode -l
```

2. Back up the original log file with ON-Bar or **ontape**.
3. After all full log files are backed up, you are prompted to switch to the next available log file and back up the new current log file.

You do not need to do the backup because you just switched to this log file.

After you free the current log file, if the log file has status **U-B** or **U-B-L**, refer to [“Freeing a Log File with Status U-B or F” on page 14-9](#) or [“Freeing a Log File with Status U-B-L.”](#)

Freeing a Log File with Status U-B-L

If a log file is backed up and all transactions within it are closed but the file is not free (status **U-B-L**), this logical-log file contains the most-recent checkpoint record.

To free log files with a status **U-B-L**, the database server must create a new checkpoint. You can execute the following command to force a checkpoint:

```
onmode -c
```

To force a checkpoint with ON-Monitor, select the **Force-Ckpt** option. ♦

UNIX

Monitoring Logging Activity

Monitor the logical-log files to determine the total available space (in all the files), the space available in the current file, and the status of a file (for example, whether the log has been backed up yet). For information on monitoring the logical-log buffers, see [“Monitoring Physical and Logical-Logging Activity” on page 16-6](#).

Monitoring the Logical Log for Fullness

You can use the following command-line utilities to monitor logical-log files.

onstat -l

The **onstat -l** utility display consists of the following three sections: physical-log information, logical-log information (general), and information on the individual logical-log files.

The section that contains the information on each logical-log file displays the following:

- The address of the logical-log file descriptor
- The log file number
- Status flags that indicate the status of each log (free, backed up, current, and so on)
- The unique ID of the log file
- The beginning page of the file
- The size of the file in pages, the number of pages used, and the percentage of pages used

To obtain the output that [Figure 14-2](#) shows:

- Use **onparams -a -i** to insert log 7 after log 1.
- Use **onmode -l** to make log 7 the current log file.
- Use **onparams -a -i** to insert log 8 after log 7.

The log file numbers (**numbers** field) can also get out of sequence if you drop several logs in the middle of the list or if the database server dynamically adds log files. For more information on **onstat -l**, see the utilities chapter in the *Administrator's Reference*.

...

address	number	flags	uniquid	begin	size	used	%used
a32eaf0	1	U-B----	7	1004ef	750	750	100.00
a32eb30	7	U---C-L	8	1027d2	750	464	61.87
a328a88	8	A-----	0	102d72	750	0	0.00
a337718	2	U-B----	2	1007dd	750	750	100.00
a337750	3	U-B----	3	100acb	750	750	100.00
a337788	4	U-B----	4	1007dd	750	750	100.00
a3377c0	5	U-B----	5	1010a7	750	750	100.00
a3377f8	6	U-B----	6	101395	750	643	85.73

...

Figure 14-2
onstat -l Output
Showing Logical-
Log File Status

oncheck -pr

The database server stores logical-log file information in the reserved pages dedicated to checkpoint information. Because the database server updates this information only during a checkpoint, it is not as recent as the information that the **onstat -l** option displays. For more details on using these options to display reserved page information, see the utilities chapter in the *Administrator's Reference*.

You can view the checkpoint reserved pages with the **oncheck -pr** command. [Figure 14-3](#) shows sample output for one of the logical-log files.

...

```
Log file number          1
Unique identifier        7
Log contains last checkpoint Page 0, byte 272
Log file flags           0x3   Log file in use
                           Current log file
Physical location        0x1004ef
Log size                 750 (p)
Number pages used        1
Date/Time file filled    01/29/2001 14:48:32
...
```

Figure 14-3
oncheck -pr Output
Containing Logical-
Log File Information

Monitoring Temporary Logical Logs

The database server uses *temporary logical logs* to roll forward transactions during a warm restore, because the permanent logs are not available then. When the roll forward completes, the database server frees the temporary log files. If you issue **onstat -l** during a warm restore, the output includes a fourth section on temporary log files in the same format as regular log files. Temporary log files use only the **B**, **C**, **F**, and **U** status flags.

Using SMI Tables

Query the **syslogs** table to obtain information on logical-log files. This table contains a row for each logical-log file. The columns are as follows.

Column	Description
number	Identification number of the logical-log file
uniqid	Unique ID of the log file
size	Size of the file in pages
used	Number of pages used
is_used	Flag that indicates whether the log file is being used
is_current	Flag that indicates whether the log file is current
is_backed_up	Flag that indicates whether the log file has been backed up
is_new	Flag that indicates whether the log file has been added since the last storage space backup
is_archived	Flag that indicates whether the log file has been written to the archive tape
is_temp	Flag that indicates whether the log file is flagged as a temporary log file

Using ON-Monitor

The **Status→Logs** option displays much of the same information for logical-log files as the **onstat -l** option displays. In addition, a column contains the dbospace in which each logical-log file is located.

Monitoring Log-Backup Status

To monitor the status of the logs and to see which logs have been backed up, use the **onstat -l** command. A status flag of **B** indicates that the log has been backed up.

Allocating Log Files

When you initialize the database server, it creates the number of logical-log files that you specify in the LOGFILES configuration parameter. These log files are the size that you specify in the LOGSIZE parameter.

Adding Logs Dynamically

The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file. When you use the default value of 2 for DYNAMIC_LOGS, the database server dynamically adds a new log file and sets off an alarm if the next active log file contains the beginning of the oldest open transaction.

The database server checks the logical-log space at these points:

- After switching to a new log file
- At the beginning of the transaction-cleanup phase of logical recovery

If the DYNAMIC_LOGS parameter is set to 1 and the next active log file contains records from an open transaction, the database server prompts you to add a log file manually and sets off an alarm. After you add the log file, the database server resumes processing the transaction.

If the DYNAMIC_LOGS parameter is set to 0 and the logical log runs out of space during a long transaction rollback, the database server can hang. (The long transaction prevents the first logical-log file from becoming free and available for reuse.) To fix the problem, set DYNAMIC_LOGS to 2 and restart the database server. Then the long transaction should be able to complete.

For more information, see [“Monitoring Events for Dynamically Added Logs” on page 14-23](#) and [“Setting High-Watermarks for Rolling Back Long Transactions” on page 14-25](#).

Size and Number of Dynamically Added Log Files

When dynamically adding a log file, the database server uses the following factors to calculate the size of the log file:

- Average log size
- Amount of contiguous space available

If the logical log is low on space, the database server adds as many log files as needed to complete the transaction. The number of log files is limited by:

- The maximum number of log files supported
- The amount of disk space for the log files
- The amount of free contiguous space in the root chunk

If the database server stops adding new log files because it is out of disk space, it writes an error message and sets off an alarm. Add a dbspace or chunk to an existing dbspace. Then the database server automatically resumes processing the transaction.

The reserve pages in the root chunk store information about each log file. The extents that contain this information expand as more log files are added. The root chunk requires two extents of 1.4 megabytes each to track 32,767 log files, the maximum number supported.

If during reversion, the chunk reserve page extent is allocated from a non-root chunk, the server attempts to put it back in the root chunk. If not enough space is available in the root chunk, the reversion fails. A message containing the required space displays in the online log. The required space needs to be freed from the root chunk before trying the reversion again.

Location of Dynamically Added Log Files

The database server allocates log files in dbspaces, in the following search order. A dbspace becomes critical if it contains logical-log files or the physical log.

Pass	Allocate Log File In
1	The dbspace that contains the newest log files (If this dbspace is full, the database server searches other dbspaces.)
2	Mirrored dbspace that contains log files (but excluding the root dbspace)
3	All dbspaces that already contain log files (excluding the root dbspace)
4	The dbspace that contains the physical log
5	The root dbspace
6	Any mirrored dbspace
7	Any dbspace

If you do not want to use this search order to allocate the new log file, you must set the DYNAMIC_LOGS parameter to 1 and execute **onparams -a -i** with the location you want to use for the new log. For details, refer to [“Monitoring Events for Dynamically Added Logs” on page 14-23](#).

Adding Logical-Log Files Manually

You might add logical-log files manually for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- To enable an open transaction to complete
- As part of moving logical-log files to a different dbspace

Warning: You cannot add a log file to a blob space or sbspace.

Add logical-log files one at a time, up to a maximum of 32,767 files, to any dbspace. As soon as you add a log file to a dbspace, it becomes a critical dbspace. You can add a logical-log file during a storage space backup.



The two ways you can add a logical-log file are:

- To the end of the file list using the **onparams -a** command or ISA
- After the current logical-log file using the **onparams -a -i** command or ISA

To add a logical-log file using onparams

1. Login as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows.
2. Ensure that the database server is in online, quiescent, or cleanup phase of fast-recovery mode.

The database server writes the following message to the log during the cleanup phase:

```
Logical recovery has reached the transaction cleanup
phase.
```

3. Decide whether you want to add the log file to the end of the log file list or after the current log file.

You can insert a log file after the current log file regardless of the DYNAMIC_LOGS parameter value. Adding a log file of a new size does not change the value of LOGSIZE.

- a. The following command adds a logical-log file to the end of the log file list in the **logspace** dbspace, using the log-file size specified by the LOGSIZE configuration parameter:

```
onparams -a -d logspace
```

- b. The following command inserts a 1000-kilobyte logical-log file after the current log file in the **logspace** dbspace:

```
onparams -a -d logspace -s 1000 -i
```

- c. To add a logical-log file with a new size (in this case, 250 kilobytes), execute the following command:

```
onparams -a -d logspace -s 250
```

4. Use **onstat -l** to check the status of the log files. The status of the new log file is **A** and is immediately available.
5. The next time you need to back up data, perform a level-0 backup of the root dbspace and the dbspaces that contain the new log files.

Although you no longer need to back up immediately after adding a log file, your next backup should be level-0 because the data structures have changed. For information on backing up data, refer to the *IBM Informix Backup and Restore Guide*.

For more information on using **onparams** to add a logical-log file, see the utilities chapter in the *Administrator's Reference*.

To add a logical-log file using ISA

1. Select **Logs→Logical** and click **Add Log File**.
2. Use **onstat -l** to check the status of the log files.
For more information, see the ISA online help.

To add a logical-log file with ON-Monitor

1. Follow the instructions on adding a log file in [“Adding Logical-Log Files Manually” on page 14-16](#), except use ON-Monitor instead of **onparams**.
2. Select **Parameters→Add-Log** to add a logical-log file.
3. In the field labelled **Dbospace Name**, enter the name of the dbospace where the new logical-log file will reside.

The size of the log file automatically appears in the **Logical Log Size** field. The new log file is always the value specified by LOGSIZE.

Dropping Logical-Log Files

To drop a logical-log file and increase the amount of the disk space available within a dbospace, you can use **onparams** or ISA. The database server requires a minimum of three logical-log files at all times. You cannot drop a log if your logical log is composed of only three log files.

The rules for dropping log files have changed:

- If you drop a log file that has never been written to (status **A**), the database server deletes it and frees the space immediately.
- If you drop a used log file (status **U-B**), the database server marks it as deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbospace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use or contains the last checkpoint record (status **C** or **L**).

To drop a logical-log file with onparams

1. Ensure that the database server is in online or quiescent mode.
2. Execute the following command to drop a logical-log file whose log file number is 21:

```
onparams -d -l 21
```

Drop log files one at a time. You must know the log file number of each logical log that you intend to drop.

3. If the log file has a status of newly Added (**A**), it is dropped immediately.
If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.
This backup prevents the database server from using the dropped log files during a restore and ensures that the reserved pages contain information about the current number of log files.

For information on using **onparams** to drop a logical-log file, see the utilities chapter in the *Administrator's Reference*.

For information on using **onlog** to display the logical-log files and unique ID numbers, see [“Displaying Logical-Log Records” on page 14-23](#).

UNIX**To drop a logical-log file with ON_Monitor**

1. Ensure that the database server is in online or quiescent mode.
2. To drop a logical-log file, select **Parameters→Drop-Log**.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.
If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.

Tip: *If the root dbspace has never been backed up, you can drop a **used** log file immediately.*



Changing the Size of Logical-Log Files

If you want to change the size of the log files, it is easier to add new log files of the desired size and then drop the old ones. You can change the size of logical-log files in the following ways:

- Use **onparams** with the **-s** option to add a new log file of a different size.
See [“Adding Logical-Log Files Manually” on page 14-16](#).
- Increase the LOGSIZE value in the ONCONFIG file if you want the database server to create larger log files.
See [“Changing Logging Configuration Parameters” on page 14-21](#).

Moving a Logical-Log File to Another Dbspace

You might want to move a logical-log file for performance reasons or to make more space in the dbspace, as explained in [“Location of Logical-Log Files” on page 13-4](#). To find out the location of logical-log files, see [“Monitoring Logging Activity” on page 14-10](#). Although moving the logical-log files is not difficult, it can be time-consuming.

Moving logical-log files is a combination of two simpler actions:

- Dropping logical-log files from their current dbspace
- Adding the logical-log files to their new dbspace

The following procedure provides an example of how to move six logical-log files from the **root** dbspace to another dbspace, **dbspace_1**.

To move the logical-log files out of the root dbspace (example)

1. Ensure that the database server is in online, quiescent, or fast-recovery mode.
2. Add six new logical-log files to **dbspace_1**.
See [“Adding Logical-Log Files Manually” on page 14-16](#).

3. Take a level-0 backup of all storage spaces to free all log files except the current log file.
(If you use **onbar -l -b -c**, you back up all log files including the current log file.) See [“Freeing a Logical-Log File” on page 14-8](#).
4. Use **onmode -l** to switch to a new current log file.
See [“Switching to the Next Logical-Log File” on page 14-7](#).
5. Drop all six logical-log files in the root dbspace.
You cannot drop the current logical-log file.
See [“Dropping Logical-Log Files” on page 14-18](#).
6. Create a level-0 backup of the root dbspace and **dbspace_1**.
For more information, see the *IBM Informix Backup and Restore Guide*.

Changing Logging Configuration Parameters

You can use a text editor or ISA to change ONCONFIG parameters. This table shows the configuration parameters for logical logs. For more information, see the chapter on configuration parameters in the *Administrator's Reference*.

Configuration Parameter	Minimum Value	Default Value	Maximum Value
DYNAMIC_LOGS	0 or 1	2	2
LOGBUFF	2 * page size	32 KB	LOGSIZE value
LOGFILES	3 files	6 files	32,767 files
LOGSIZE	1500 KB on UNIX 500 KB on Windows	2000 KB	See the <i>Administrator's Reference</i>
LTXEHWM	LTXHWM value	90%	100%
LTXHWM	1%	80%	100%



Important: The change to LOGFILES does not take effect until you reinitialize the disk space.

To change the logical-log configuration parameters in the ONCONFIG file

1. Bring the database server offline or into quiescent mode.
2. Use ISA or a text editor to update the configuration parameters.
The DYNAMIC_LOGS, LTXHWM, and LTXEHWM parameters are not in the **onconfig.std** file. To change the values of these parameters, add them to your ONCONFIG file.
3. Shut down and restart the database server.
4. Perform this step only if you are changing LOGFILES.
 - a. Unload all the database server data.
You cannot rely on storage space backups to unload and restore the data because a restore returns the parameters to their previous value.
 - b. Reinitialize the database server disk space.
For more information, see [“Initializing Disk Space” on page 4-4](#).
 - c. Re-create all databases and tables.
 - d. Reload all the database server data.
For information on loading and unloading data, see the *IBM Informix Migration Guide*.
5. Back up the root dbspace to enable your changed logical logs.

UNIX

Using ON-Monitor to Change LOGFILES

You can use ON-Monitor to change the values of LOGFILES.

To change these values

1. Unload all the database server data.
You cannot rely on storage space backups to unload and restore the data because a restore returns the parameters to their previous value.
2. Select **Parameters→Initialize** to reinitialize disk space.
3. Change the value of LOGSIZE in the field labelled **Log.Log Size**, or change the value of LOGFILES in the field labelled **Number of Logical Logs**.
4. Proceed with the database server disk-space initialization.

5. Re-create all databases and tables.
6. Reload all the database server data.

For information on loading and unloading data, see the *IBM Informix Migration Guide*.

Displaying Logical-Log Records

Use the **onlog** utility to display and interpret logical-log records. For information on using **onlog**, see the utilities chapter in the *Administrator's Reference*.

Monitoring Events for Dynamically Added Logs

Monitor the following event alarms that dynamically added log files trigger (see [Figure 14-4](#)). When each alarm is triggered, a message is written to the message log. For more information, see the chapters on event alarms and configuration parameters in the *Administrator's Reference*.

You can include the **onparams** command to add log files in your alarm script for event class ID 27, log file required. Your script can also execute the **onstat -d** command to check for adequate space and execute the **onparams a -i** command with the location that has enough space. You must use the **-i** option to add the new log right after the current log file.

Figure 14-4
Event Alarms for Dynamically Added Log Files

Class ID	Severity	Class Message	Message
26	3	Dynamically added log file <i>log_number</i>	<p>This message displays when the database server dynamically adds a log file.</p> <p>Dynamically added log file <i>log_number</i> to DBspace <i>dbspace_number</i>.</p>
27	4	Log file required	<p>This message displays when DYNAMIC_LOGS is set to 1 and the database server is waiting for you to add a log file.</p> <p>ALERT: The oldest logical log <i>log_number</i> contains records from an open transaction <i>transaction_address</i>. Logical logging will remain blocked until a log file is added. Add the log file with the onparams -a command, using the -i (insert) option, as in:</p> <pre>onparams -a -d <i>dbspace</i> -s <i>size</i> -i</pre> <p>Then complete the transaction as soon as possible.</p>
28	4	No space for log file	<p>ALERT: Because the oldest logical log <i>log_number</i> contains records from an open transaction <i>transaction_address</i>, the server is attempting to dynamically add a log file. But there is no space available. Please add a DBspace or chunk. Then complete the transaction as soon as possible.</p>

Figure 14-5 shows the actions that the database server takes for each setting of the DYNAMIC_LOGS configuration parameter.

Figure 14-5
DYNAMIC_LOGS Settings

DYNAMIC_LOGS	Meaning	Event Alarm	Wait to Add Log	Dynamic Log Add
2 (default)	Allows automatic allocation of new log files to prevent open transactions from hanging the system.	Yes (26, 28)	No	Yes
1	Allows manual addition of new log files.	Yes (27)	Yes	No
0	Does not allocate log files but issues the following message about open transactions: WARNING: The oldest logical-log file <i>log_number</i> contains records from an open transaction <i>transaction_address</i> , but the dynamic log feature is turned off.	No	No	No

Setting High-Watermarks for Rolling Back Long Transactions

The database server uses the LTXHWM and LTXEHWM configuration parameters to set high-watermarks for long transactions. If DYNAMIC_LOGS is set to 1 or 2, the default LTXHWM value is 80 percent and LTXEHWM is 90 percent. If DYNAMIC_LOGS is set to 0, the default LTXHWM value is 50 percent and the default LTXHEWM value is 60 percent.

If you decrease your high-watermark values, you increase the likelihood of long transactions. To compensate, allocate additional log space. For information on LTXHWM and LTXEHWM, see the chapter on configuration parameters in the *Administrator's Reference*.

Long-Transaction High-Watermark (LTXHWM)

The *long-transaction high-watermark* is the percentage of total log space that a transaction is allowed to span before it is rolled back. If the database server finds an open transaction in the oldest used log file, it dynamically adds log files. Because the log space is increasing, the high-watermark expands outward. When the log space reaches the high-watermark, the database server rolls back the transaction. The transaction rollback and other processes also generate logical-log records. The database server continues adding log files until the rollback is complete to prevent the logical log from running out of space. More than one transaction can be rolled back if more than one long transaction exists.

For example, the database server has ten logical logs and LTXHWM is set to 98. A transaction begins in log file 1 and update activity fills logs 1 through 9. The database server dynamically adds log file 11 after log file 10. As long as the transaction does not complete, this process continues until the database server has added 40 log files. When the database server adds the fiftieth log, the transaction has caught up to the high-watermark and the database server rolls it back.

Exclusive Access, Long-Transaction High-Watermark (LTXEHW)

The *exclusive-access, long-transaction high-watermark* occurs when the long transaction currently being rolled back is given *exclusive* access to the logical log. The database server dramatically reduces log-record generation. Only threads that are currently rolling back transactions and threads that are currently writing COMMIT records are allowed access to the logical log. Restricting access to the logical log preserves as much space as possible for rollback records that are being written by the user threads that are rolling back transactions.



Warning: If you set both LTXHWM and LTXEHW to 100, long transactions are never aborted. It is recommended that you set LTXHWM to below 100 for normal database server operations. Set LTXHWM to 100 to run scheduled transactions of unknown length. Set LTXEHW to 100 if you never want to block other users while a long transaction is rolling back and you have ample disk space.

Adjusting the Size of Log Files to Prevent Long Transactions

Use larger log files when many users are writing to the logs at the same time. If you use small logs and long transactions are likely to occur, reduce the high-watermark. Set the LTXHWM value to 50 and the LTXEHW value to 60.

If the log files are too small, the database server might run out of log space while rolling back a long transaction. In this case, the database server cannot block fast enough to add a new log file before the last one fills. If the last log file fills, the system will hang and display an error message. To fix the problem, shut down and restart the database server. For details, see [“Recovering from a Long Transaction Hang” on page 14-27](#).

Recovering from a Long Transaction Hang

If your system has ample disk space and you want to perform transactions of unknown length, consider setting LTXHWM to 100 to force the database server to continue adding log files until you complete the transaction.

A transaction might hang because the database server has run out of disk space. The database server stops adding new log files, writes an error message, and sets off an alarm.

To continue the transaction

1. To continue the transaction, add a dbspace or chunk to a dbspace.
2. Resume processing the transaction.

If you cannot add more disk space to the database server, abort the transaction.

To abort the transaction

- Issue the **onmode -z** command.
- Shut down and restart the database server.

When the database server comes up in fast-recovery mode, the transaction is rolled back. Then perform the following steps:

To recover from a long transaction hang

- 1.** Add more disk space or another disk until the transaction is successfully rolled back.
- 2.** Perform a point-in time restore to a time before the long transaction began or early enough for the database server to roll back the transaction.
- 3.** Drop the extra log files, dbspaces, or chunks from the database server instance.
- 4.** Perform a complete level-0 backup to free the logical-log space.

Physical Logging, Checkpoints, and Fast Recovery

In This Chapter	15-3
Critical Sections	15-3
Physical Logging	15-4
Fast Recovery Use of Physically-Logged Pages	15-4
Backup Use of Physically-Logged Pages	15-4
Database Server Activity That Is Physically Logged	15-5
Physical Recovery Messages	15-5
Physical Logging and Simple Large Objects	15-6
Physical Logging and Smart Large Objects	15-6
Size and Location of the Physical Log	15-6
Specifying the Location of the Physical Log	15-6
Estimating the Size of the Physical Log	15-7
Configuring the Size of the Physical Log	15-8
Physical-Log Overflow When Many Users Are in Critical Sections	15-8
Effect of Checkpoints on the Physical-Log Size	15-8
Physical-Log Overflow When Transaction Logging Is Turned Off	15-9
Details of Physical Logging	15-9
Page Is Read into the Shared-Memory Buffer Pool.	15-10
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	15-10
Change Is Reflected in the Data Buffer	15-10
Physical-Log Buffer Is Flushed to the Physical Log	15-10
Page Buffer Is Flushed	15-11
When a Checkpoint Occurs	15-11
How the Physical Log Is Emptied	15-11

Checkpoints	15-11
Full Checkpoint.	15-12
Fuzzy Checkpoint	15-12
Fuzzy Operations.	15-12
Write-Ahead Logging and Fast Recovery.	15-13
Fuzzy Checkpoints Improve Performance	15-13
Events That Initiate a Fuzzy Checkpoint	15-14
Events That Initiate a Full Checkpoint	15-14
Sequence of Events in a Checkpoint	15-15
User Threads Cannot Enter a Critical Section	15-16
Logical-Log Buffer Is Flushed to the Logical-Log File on Disk.	15-16
The Physical-Log Buffer is Flushed to the Physical Log on Disk.	15-16
Modified Pages in the Buffer Pool Are Flushed to Disk	15-17
Checkpoint Thread Writes Checkpoint Record	15-17
Physical Log Is Logically Emptied	15-17
Backup and Restore Considerations.	15-18
Fast Recovery	15-18
Need for Fast Recovery	15-18
Situations When Fast Recovery Is Initiated	15-19
Fast Recovery and Buffered Logging	15-19
Possible Physical Log Overflow During Fast Recovery	15-19
Fast Recovery and No Logging	15-19
Details of Fast Recovery After A Full Checkpoint	15-20
Returning to the Last-Checkpoint State	15-21
Finding the Checkpoint Record in the Logical Log	15-21
Rolling Forward Logical-Log Records.	15-22
Rolling Back Incomplete Transactions.	15-22
Details of Fast Recovery After A Fuzzy Checkpoint	15-23
Returning to the Last-Checkpoint State for Nonfuzzy Operations	15-24
Locating the Oldest Update in the Logical Log.	15-25
Applying the Log Records for Fuzzy Operations	15-26
Rolling Forward Logical-Log Records.	15-26
Rolling Back Incomplete Transactions.	15-28

In This Chapter

This chapter covers the three procedures that the database server uses to achieve data consistency:

- Physical logging
- Checkpoints
- Fast recovery

The *physical log* is a set of disk pages where the database server stores an unmodified copy of the page called a *before-image*. *Physical logging* is the process of storing a before-image of a page that the database server is going to change. A *checkpoint* refers to a point when the database server synchronizes the pages on disk with the pages in the shared-memory buffers. *Fast recovery* is an automatic procedure that restores the database server to a consistent state after it goes offline under uncontrolled conditions.

These procedures ensure that multiple, logically related writes are recorded as a unit, and that data in shared memory is periodically made consistent with data on disk.

For the tasks to manage and monitor the physical log and checkpoints, see [Chapter 16, “Managing the Physical Log.”](#)

Critical Sections

A *critical section* is a section of code (or machine instructions) that must be performed as a single unit. A critical section ensures the integrity of a thread by allowing it to execute a series of instructions before it is swapped out.

Physical Logging

Physical logging is the process of storing the pages that the database server is going to change before the changed pages are actually recorded on disk. Before the database server modifies certain pages in the shared-memory buffer pool, it stores before-images of the pages in the physical-log buffer in shared memory.

The database server maintains the before-image page in the physical-log buffer in shared memory for those pages until one or more page cleaners flush the pages to disk. The unmodified pages are available in case the database server fails or the backup procedure needs them to provide an accurate snapshot of the database server data. Fast recovery and database server backups use these snapshots.

The database server empties the physical log at each checkpoint (except in the special circumstances explained in [“Configuring the Size of the Physical Log” on page 15-8](#)). For more information on checkpoints, see [“Checkpoints” on page 15-11](#).

Fast Recovery Use of Physically-Logged Pages

After a failure, the database server uses the before-images of pages modified by non-fuzzy operations to restore these pages on the disk to their state at the last checkpoint. Then the database server uses the logical-log records to return all data to physical and logical consistency, up to the point of the most-recently completed transaction. [“Fast Recovery” on page 15-18](#) explains this procedure in more detail.

Backup Use of Physically-Logged Pages

When you perform a backup, the database server performs a checkpoint and uses the physical log to find the changed pages. In a level-0 backup, the database server backs up all disk pages. In a level-1 or level-2 backup, the database server backs up only the changed pages. For more details, see the *IBM Informix Backup and Restore Guide*.

Database Server Activity That Is Physically Logged

In case of multiple modifications before the next checkpoint, only the first before-image is logged in the physical log.

The database server stores the before-images in the physical log only until the next checkpoint. To control the amount of data that the database server logs, you can tune the checkpoint interval configuration parameter CKPTINTVL.

The following dbspace page modifications are *not* physically logged:

- Pages that do not have a valid database server address
This situation usually occurs when the page was used by some other database server or a table that was dropped.
- Pages that the database server has not used and that are located in a dbspace where no table has been dropped since the last checkpoint
- Pages for fuzzy operations



Important: The database server no longer logs the before-images for fuzzy operations in the physical log. It still tracks these updates in the logical log. For a definition of fuzzy operations, see *“Fuzzy Operations” on page 15-12*.

Physical Recovery Messages

When fast recovery begins, the database server logs the following message:

```
Physical recovery started at page chunk:offset.
```

When the fast recovery completes, the database server logs the following message:

```
Physical recovery complete: number pages examined, number pages restored.
```

If the *number of pages examined* is much larger than the *number of pages restored*, increase the size of the buffer pool to reduce the number of duplicate before-images. For more information, see the messages appendix in the *Administrator's Reference*.

Physical Logging and Simple Large Objects

The database server pages in the physical log can be any database server page, including simple large objects in tblspaces. Even overhead pages (such as chunk free-list pages) are copied to the physical log before data on the page is modified and flushed to disk.

Overhead pages also include blobspace free-map pages and blobspace bit-map pages. Blobspace blobpages are not logged in the physical log. For further information about blobspace logging, see [“Logging Blobspaces and Simple Large Objects” on page 13-11](#).

Physical Logging and Smart Large Objects

The user-data portion of smart large objects is not physically logged. However, the metadata is physically logged. For information on smart large objects, see [“Sbspaces” on page 9-21](#).

Size and Location of the Physical Log

This section describes how to configure the size and location of the physical log.

Specifying the Location of the Physical Log

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no initial control over this placement. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize disk contention), you can move the physical log out of the root dbspace to another dbspace, preferably to a disk that does not contain active tables or the logical-log files.

The physical log is located in the dbspace specified by the ONCONFIG parameter PHYSDBS. (For information on PHYSDBS, see the chapter on configuration parameters in the *Administrator's Reference*.) Change PHYSDBS only if you decide to move the physical-log file from the root dbspace. (See [“Changing the Physical-Log Location and Size” on page 16-3](#).)

Because the physical log is critical, it is recommended that you mirror the dbspace that contains the physical log.

Estimating the Size of the Physical Log

If your workload is update intensive, you can use the following formula to calculate the size of the physical log where PHYSFILE equals the physical-log size:

$$\text{PHYSFILE} = (\text{users} * \text{max_log_pages_per_crit_sect} * 4 * \text{pagesize}) / 1024$$

Variable in Formula	Description
<i>users</i>	The maximum number of concurrent user threads for which you can obtain an estimate when you execute onstat -u during peak processing. The last line of the onstat -u output contains the maximum number of concurrent user threads. If you set the NETTYPE parameter, sum the values specified in the <i>users</i> field of each NETTYPE parameter in your ONCONFIG file.
<i>max_log_pages_per_crit_sect</i>	Maximum number of pages that the database server can physically log in a critical section. Substitute one of the following values: 5 if you do not use R-tree indexes 10 if you use R-tree indexes
4	Necessary factor because this part of the formula represents only 25 percent of the physical log: <i>users * max_log_pages_per_crit_sect</i>
<i>pagesize</i>	System page size in bytes that you can obtain with oncheck -pr
1024	Necessary divisor, because you specify the PHYSFILE parameter in units of kilobytes

This formula is based on how much physical logging space the database server needs in a worst-case scenario. This scenario takes place when a check-point occurs because the log becomes 75 percent full. This size might be too small or large for your actual workload or configuration.

If you are using simple large objects in a dbspace in a database without logging, substitute the size of the most-frequently occurring simple large object in the dbspace for the maximum log pages per critical section.

For more information on monitoring and tuning the physical log, refer to the chapter on configuration effects on I/O utilization in your *Performance Guide*.

Configuring the Size of the Physical Log

Because a checkpoint logically empties the physical log when it becomes 75 percent full, it is unlikely that the log would become 100 percent full before the checkpoint completes. To assure further that the physical log does not become full during a checkpoint, take the following actions:

- Configure the database server according to the sizing guidelines for the physical log and the logical-log files.
- Fine-tune the size of the physical log by monitoring it during production activity.

Fuzzy checkpoints keep the physical log from filling up too quickly when applications are doing intensive updates. (See [“Fuzzy Checkpoint” on page 15-12](#).) However, the physical log could still become full, as the following sections describe.

Physical-Log Overflow When Many Users Are in Critical Sections

A checkpoint cannot occur if any thread is in a critical section. (See [“Critical Sections” on page 15-3](#).) This scenario can occur when the thread that does the checkpoint also rolls back a long transaction.

Effect of Checkpoints on the Physical-Log Size

Fuzzy checkpoints keep the physical log from filling up too quickly when applications are doing intensive updates. You can reduce the size of the physical log when applications require less intensive updates or when updates tend to cluster within the same pages. You can decrease the size of the physical log if you intend to use physical-log fullness to trigger checkpoints.

If you increase the checkpoint interval or anticipate increased activity, consider increasing the size of the physical log. For more information, see the chapter on effects of configuration on I/O activity in your *Performance Guide*.

Physical-Log Overflow When Transaction Logging Is Turned Off

The physical log can overflow if you use simple large objects or smart large objects in a database with transaction logging turned off, as the following example shows.

When the database server processes these simple large objects, each portion of the simple large object that the database server stores on disk can be logged separately, allowing the thread to exit the critical sections of code between each portion. However, if logging is turned off, the database server must carry out all operations on the simple large object in one critical section. If the simple large object is large and the physical log small, this scenario can cause the physical log to become full. If this situation occurs, the database server sends the following message to the message log:

```
Physical log file overflow
```

The database server then initiates a shutdown. For the suggested corrective action, refer to this message in your message log.

Details of Physical Logging

This section describes the details of physical logging. It is provided to satisfy your curiosity; you do not need to understand the information here in order to manage your physical log.

The database server performs physical logging in the following six steps:

1. Reads the data page from disk to the shared-memory page buffer (if the data page is not there already)
2. Copies the unchanged page to the physical-log buffer
3. Reflects the change in the page buffer after an application modifies data
4. Flushes the physical-log buffer to the physical log on disk

5. Flushes the page buffer and writes it back to disk
6. When a checkpoint occurs, flushes the physical-log buffer to the physical log on disk and empties the physical log

The paragraphs that follow explain each step in detail.

Page Is Read into the Shared-Memory Buffer Pool

When a session requests a row, the database server identifies the page on which the row resides and attempts to locate the page in the database server shared-memory buffer pool. If the page is not already in shared memory, it is read into the resident portion of the database server shared memory from disk.

A Copy of the Page Buffer Is Stored in the Physical-Log Buffer

If the before-image of a modified page is stored in the physical-log buffer, it is eventually flushed from the physical-log buffer to the physical log on disk. The before-image of the page plays a critical role in restoring data and fast recovery. For more details, see [“Physical-Log Buffer” on page 7-20](#).

Change Is Reflected in the Data Buffer

The database server reflects changes to the data in the shared-memory data buffer. Data from the application is passed to the database server. After a copy of the unchanged data page is stored in the physical-log buffer, the new data is written to the page buffer already acquired.

Physical-Log Buffer Is Flushed to the Physical Log

The database server flushes the physical-log buffer before it flushes the data buffer to ensure that a copy of the unchanged page is available until the changed page is copied to disk. The before-image of the page is no longer needed after a checkpoint occurs. For more details, see [“Flushing the Physical-Log Buffer” on page 7-42](#).

Page Buffer Is Flushed

After the physical-log buffer is flushed, the shared-memory page buffer is flushed to disk (such as during a checkpoint), and the data page is written to disk. Only non-fuzzy pages are flushed to disk during a fuzzy checkpoint. For conditions that lead to the flushing of the page buffer, see [“Flushing Data to Disk” on page 7-40](#).

When a Checkpoint Occurs

A checkpoint can occur at any point in the physical-logging process. The database server performs two types of checkpoints: *full* and *fuzzy*. For information, see [“Checkpoints” on page 15-11](#).

How the Physical Log Is Emptied

The database server manages the physical log as a circular file, constantly overwriting unneeded data. The checkpoint procedure empties the physical log by resetting a pointer in the physical log that marks the beginning of the next group of required before-images.

Checkpoints

The database server performs two types of checkpoints: full checkpoints (also known as *sync* checkpoints) and fuzzy checkpoints. The term *checkpoint* refers to the point in the database server operation when the pages on disk are synchronized with the pages in the shared-memory buffer pool. The default checkpoint type is fuzzy.

The database server generates at least one checkpoint for each span of the logical-log space to guarantee that it has a checkpoint at which to begin fast recovery.

Although the database server performs checkpoints automatically, you can initiate one manually or control how often the database server checks to see if a checkpoint is needed. You can specify the checkpoint interval in the CKPTINTVL configuration parameter. To reduce the amount of work required at checkpoint, lower the LRU_MAX and LRU_MIN values. For instance, if the database server has several instances with a very large buffer pool, set the LRU_MAX_DIRTY value to less than 1 to reduce the checkpoint time required. For more information about CKPTINTVL, LRU_MAX, and LRU_MIN, see the chapter on configuration parameters in the *Administrator's Reference*. For information on monitoring and tuning checkpoint parameters, see your *Performance Guide*.

Full Checkpoint

In a *full checkpoint*, the database server flushes all modified pages in the shared-memory buffer pool to disk. When a full checkpoint completes, all physical operations are complete, the MLRU queue is empty, and the database server is said to be physically consistent.

Fuzzy Checkpoint

In a *fuzzy checkpoint*, the database server does not flush the modified pages in the shared-memory buffer pool to disk for certain types of operations, called *fuzzy operations*. When a fuzzy checkpoint completes, the pages might not be consistent with each other, because the database server does not flush all data pages to disk. A fuzzy checkpoint completes much more quickly than a full checkpoint and reduces the amount of physical logging during heavy update activity. When necessary, the database server performs a full checkpoint to ensure the physical consistency of all data on disk.

Fuzzy Operations

The following commonly used operations are *fuzzy* for built-in data types:

- Inserts
- Updates
- Deletes

The following operations are *nonfuzzy*:

- Inserts, updates, and deletes for rows that contain user-defined data types, smart large objects (CLOB and BLOB data types), or simple large objects (TEXT and BYTE data types)
- Table alters and loads
- Operations that create or modify indexes (B-tree, R-tree, or user-defined indexes)

The database server flushes *all* the modified data pages for nonfuzzy operations to disk during a fuzzy checkpoint in the same way as for a full checkpoint.



Important: Fuzzy checkpoints are disabled for the primary and secondary servers in a High-Availability Data Replication pair.

Write-Ahead Logging and Fast Recovery

Fuzzy checkpoint uses write-ahead logging for fast recovery. *Write-ahead logging* means that the logical-log records representing changes to fuzzy data must be on disk before the changed data replaces the previous version of the data on disk. Fast recovery begins with the oldest update not yet flushed to disk rather than with the previous checkpoint.

Fuzzy checkpoints result in slightly longer roll-forward fast-recovery times. The database server occasionally performs a full checkpoint to prevent loss of old logical-log records.

Fuzzy Checkpoints Improve Performance

Fuzzy checkpoints are much faster than full checkpoints and improve transaction throughput. Because the database server does not log fuzzy operations in the physical log, the physical log does not fill as quickly, and checkpoints occur less often. For example, if you are inserting and updating a lot of data, checkpoints occur less frequently and are shorter.

The database server skips a full checkpoint if all data is physically consistent when the checkpoint interval expires. It skips a fuzzy checkpoint only if no pages have been dirtied since the last checkpoint.

To improve transaction throughput, increase the LRU_MAX_DIRTY and LRU_MIN_DIRTY values. Do not change the gap between the LRU_MAX_DIRTY and LRU_MIN_DIRTY values. For information on improving checkpoint performance, see the chapter about configuration impacts on I/O in your *Performance Guide*.

Events That Initiate a Fuzzy Checkpoint

Usually, when the database server performs an automatic checkpoint, it is a fuzzy checkpoint. Any one of following conditions initiates a fuzzy checkpoint:

- The checkpoint interval, specified by the configuration parameter CKPTINTVL, has elapsed, and one or more modifications have occurred since the last checkpoint.
- The physical log on disk becomes 75 percent full.
- The database server detects that the next logical-log file to become current contains the most recent checkpoint record.
- Certain administrative tasks, such as adding a chunk or a dbspace, take place.

Events That Initiate a Full Checkpoint

In the following situations, the database server performs a full checkpoint to ensure the physical consistency of all data on disk:

- When you issue **onmode -ky** to shut down the database server
- When you force a checkpoint using **onmode -c** or ISA
For more information, see [“Forcing a Full Checkpoint” on page 16-9](#).
- When you convert the database server to a newer version or revert to a previous version
- When you perform a backup or restore using ON-Bar or **ontape**
The backup tool performs a full checkpoint automatically to ensure the physical consistency of all data before it writes it to the backup media.
- At the end of fast recovery or full recovery

- If the database server is about to switch to the next free log and the log following the free log contains the oldest update

For example, suppose four logical-log files have the status shown in the following list. The database server forces a full checkpoint when it switches to logical-log file 3 if the logical-log file 4 has the oldest update. The full checkpoint advances the oldest update to logical-log file 3.

logid	Logical-Log File Status
1	U-B----
2	U---C--
3	F
4	U-B---L

The database server performs a full checkpoint to prevent problems with fast recovery of old log records.

For a list of situations in which you should initiate a full checkpoint, see the following section.

Sequence of Events in a Checkpoint

The following section outlines the main events that occur from the time a checkpoint is requested. This section also notes the differences between full and fuzzy checkpoints:

1. The database server prevents user threads from entering critical sections.
2. The logical-log buffer is flushed to the current logical-log file on disk.
3. The database server flushes the physical-log buffer to the physical log.
4. In a fuzzy checkpoint, the database server flushes modified pages for nonfuzzy operations to disk.

In a full checkpoint, the database server flushes all modified pages to disk.

5. The database server writes a checkpoint record to the logical-log buffer.
6. The physical log is logically emptied. (Current entries can be overwritten).

User Threads Cannot Enter a Critical Section

This step is the same for both fuzzy and full checkpoints. Once a checkpoint is requested, user threads are prevented from entering portions of code that are considered critical sections. User threads that are within critical sections of code are permitted to continue processing to the end of the critical sections.

Logical-Log Buffer Is Flushed to the Logical-Log File on Disk

This step is the same for both fuzzy and full checkpoints. Next, the logical-log buffer is flushed to the logical-log file on disk.

The Physical-Log Buffer is Flushed to the Physical Log on Disk

For both fuzzy and full checkpoints, the database server flushes the physical-log buffer to the physical log on disk. For full checkpoints, the physical-log buffer contains all modified pages whereas for fuzzy checkpoints, the physical-log buffer holds pages that fuzzy operations modified. The time stamp of the physical-log flush is stored in shared memory.

Modified Pages in the Buffer Pool Are Flushed to Disk

In a fuzzy checkpoint, the database server flushes modified pages for nonfuzzy operations in the buffer pool to disk. The database server flushes modified pages for fuzzy operations (inserts, deletes, updates) to disk.

Figure 15-6 shows how the database server writes only the nonfuzzy pages to disk. The shaded squares, marked F, represent the fuzzy pages.

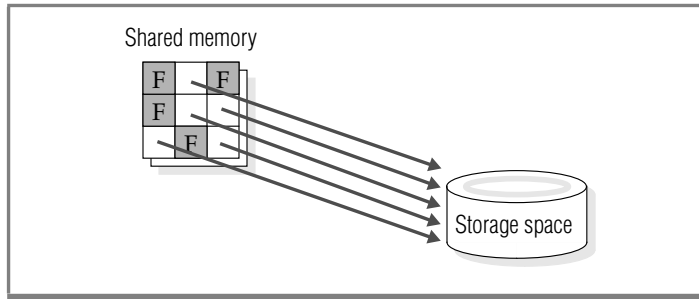


Figure 15-6
*Selectively Writing
Modified Pages
from Shared
Memory to Disk*

In a full checkpoint, the database server flushes all modified pages in the shared-memory buffer pool to disk.

Checkpoint Thread Writes Checkpoint Record

This step is the same for both fuzzy and full checkpoints. A *checkpoint-complete* record is written to the logical-log buffer after the modified pages have been written to disk.

In a fuzzy checkpoint, a dirty-pages table (DPT) record is written to the logical-log buffer. For more information, see the chapter on logical-log record types in the *Administrator's Reference*.

Physical Log Is Logically Emptied

This step is the same for both fuzzy and full checkpoints. After the checkpoint-complete record is written to disk, the physical log is logically emptied, meaning that current entries in the physical log can be overwritten.



Backup and Restore Considerations

If you perform a backup, the database server performs a full checkpoint and flushes all changed pages, including those for fuzzy operations, to the disk. If you perform a restore, the database server reapplies all logical-log records.

Important: *Because the logical log contains records of fuzzy operations not yet written to disk, you must back up the logical logs regularly.*

For information on ON-Bar or **ontape**, see the *IBM Informix Backup and Restore Guide*.

Fast Recovery

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time that it moves from offline to quiescent mode or from offline to online mode. You do not need to take any administrative actions for fast recovery; it is an automatic feature.

The fast-recovery process checks if, the last time that the database server went offline, it did so in uncontrolled conditions. If so, fast recovery returns the database server to a state of physical and logical consistency, as described in [“Details of Fast Recovery After A Full Checkpoint” on page 15-20](#).

If the fast-recovery process finds that the database server came offline in a controlled manner, the fast-recovery process terminates, and the database server moves to online mode.

Need for Fast Recovery

Fast recovery restores the database server to physical and logical consistency after any failure that results in the loss of the contents of memory for the database server. For example, the operating system fails without warning. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Situations When Fast Recovery Is Initiated

Every time that the administrator brings the database server to quiescent mode or online mode from offline mode, the database server checks to see if fast recovery is needed.

As part of shared-memory initialization, the database server checks the contents of the physical log. The physical log is empty when the database server shuts down under control. The move from online mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if the database server finds pages in the physical log, the database server clearly went offline under uncontrolled conditions, and fast recovery begins.

Fast Recovery and Buffered Logging

If a database uses buffered logging (as described in [“Buffered Transaction Logging” on page 11-12](#)), some logical-log records associated with committed transactions might not be written to the logical log at the time of the failure. If this occurs, fast recovery cannot restore those transactions. Fast recovery can restore only transactions with an associated COMMIT record stored in the logical log on disk. (For this reason, buffered logging represents a trade-off between performance and data vulnerability.)

Possible Physical Log Overflow During Fast Recovery

During fast recovery, the physical log can overflow. When this occurs, the server tries to extend the physical log space to a disk file. The default location of this file is `$INFORMIXDIR/tmp`. The `ONCONFIG` parameter `PLOG_OVERFLOW_PATH` is used to define the location for creating this file. The file is named `plog_extend.servernum`. This extended physical log file is removed by the server when the first checkpoint is performed during a fast recovery.

Fast Recovery and No Logging

For databases or tables that do not use logging, fast recovery restores the database to its state at the time of the most recent checkpoint. All changes made to the database since the last checkpoint are lost. All fuzzy operations (inserts, deletes, updates) not yet flushed to disk are also lost.

Details of Fast Recovery After A Full Checkpoint

Fast recovery works differently depending on whether the previous checkpoint was a full or fuzzy checkpoint. This section discusses fast recovery after a full checkpoint.

Fast recovery returns the database server to a consistent state as part of shared-memory initialization. The consistent state means that all committed transactions are restored, and all uncommitted transactions are rolled back.

Fast recovery is accomplished in the following two stages:

- The database server uses the physical log to return to the most recent point of known *physical consistency*, the most recent checkpoint.
- The database server uses the logical-log files to return to *logical consistency* by rolling forward all committed transactions that occurred after the last checkpoint and rolling back all transactions that were left incomplete.

Fast recovery occurs in the following steps:

1. Uses the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint.
2. Locates the most recent checkpoint record in the logical-log files.
3. Rolls forward all logical-log records written after the most recent checkpoint record.
4. Rolls back transactions that do not have an associated COMMIT or BEGCOM record in the logical log.

The database server writes the BEGCOM record when a transaction commits. For details, see the chapter on logical-log records in the *Administrator's Reference*.

The paragraphs that follow describe each step in detail.

Returning to the Last-Checkpoint State

To accomplish the first step, returning all disk pages to their condition at the time of the most recent checkpoint, the database server writes the before-images stored in the physical log to shared memory and then back to disk. Each before-image in the physical log contains the address of a page that was updated after the checkpoint. When the database server writes each before-image page in the physical log to shared memory and then back to disk, changes to the database server data since the time of the most recent checkpoint are undone. Figure 15-7 illustrates this step.

The database server is now physically consistent.

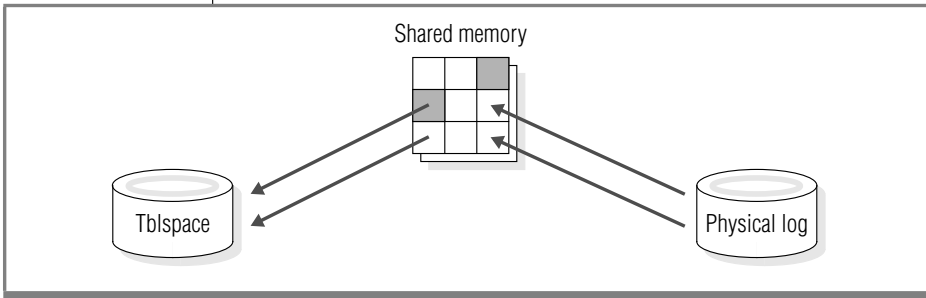


Figure 15-7
*Writing All
Remaining Before-
Images in the
Physical Log Back
to Disk*

Finding the Checkpoint Record in the Logical Log

In the second step, the database server locates the address of the most recent checkpoint record in the logical log. The most recent checkpoint record is guaranteed to be in the logical log on disk.

Rolling Forward Logical-Log Records

The third step in fast recovery rolls forward the logical-log records that were written after the most recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point at which the uncontrolled shutdown occurred. [Figure 15-8](#) illustrates this step.

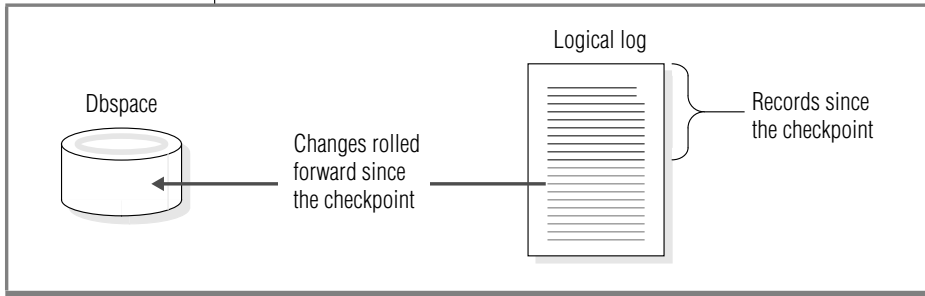


Figure 15-8
*Rolling Forward the
Logical-Log
Records Written
Since the Most
Recent Checkpoint*

Rolling Back Incomplete Transactions

The final step in fast recovery rolls back all logical-log records for transactions that were not committed at the time the system failed. All databases are logically consistent because all committed transactions are rolled forward and all uncommitted transactions are rolled back.

Transactions that have completed the first phase of a two-phase commit are exceptional cases. For more information, see [“How the Two-Phase Commit Protocol Handles Failures”](#) on page 22-9.

Because one or more transactions possibly spanned several checkpoints without being committed, this rollback procedure might read backward through the logical log past the most recent checkpoint record. All logical-log files that contain records for open transactions are available to the database server because a log file is not freed until all transactions that it contains are closed. Figure 15-9 illustrates the rollback procedure. When fast recovery is complete, the database server goes to quiescent or online mode.

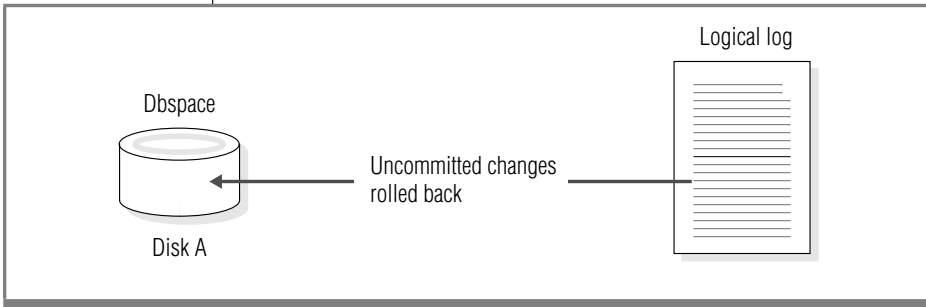


Figure 15-9
*Rolling Back All
Incomplete
Transactions*

Details of Fast Recovery After A Fuzzy Checkpoint

This section discusses fast recovery after a fuzzy checkpoint. Fast recovery is accomplished in the following stages:

- The database server uses the physical log to return to the most recent checkpoint. The database server might not be physically consistent at this point in fast recovery because fuzzy operations do not physically log the before-image of pages.
- The database server processes the logical-log records starting with the oldest update that has not yet been flushed to disk, rather than starting with the previous checkpoint.
- The database server uses the logical-log files to return to *logical consistency* by rolling forward all committed transactions that occurred after the last checkpoint and rolling back all transactions that were left incomplete.

The following steps describe these stages in detail:

1. Uses the data in the physical log to return disk pages for nonfuzzy operations to their condition at the time of the most recent checkpoint.
2. Locates the oldest update in the logical log that is not yet flushed to disk.
3. Applies the log records for fuzzy operations that occurred before the most recent checkpoint.
4. Rolls forward all logical-log records written after the most recent checkpoint record.
5. Rolls back transactions that do not have an associated COMMIT or BEGIN COMMIT record in the logical log.

Although fast recovery after a fuzzy checkpoint takes longer than after a full checkpoint, you can optimize it. For details, see your *Performance Guide*.

Returning to the Last-Checkpoint State for Nonfuzzy Operations

To accomplish the first step and return all disk pages for nonfuzzy operations to their condition at the time of the most recent checkpoint, the database server writes the before-images stored in the physical log to shared memory and then back to disk. Each before-image in the physical log contains the address of a page that was updated after the checkpoint. When the database server writes each before-image page in the physical log to shared memory and then back to disk, changes to the database server data since the time of the most recent checkpoint are undone. [Figure 15-10](#) illustrates this step.

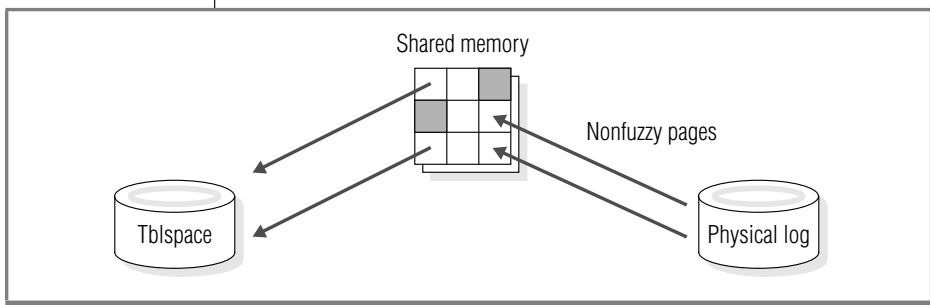


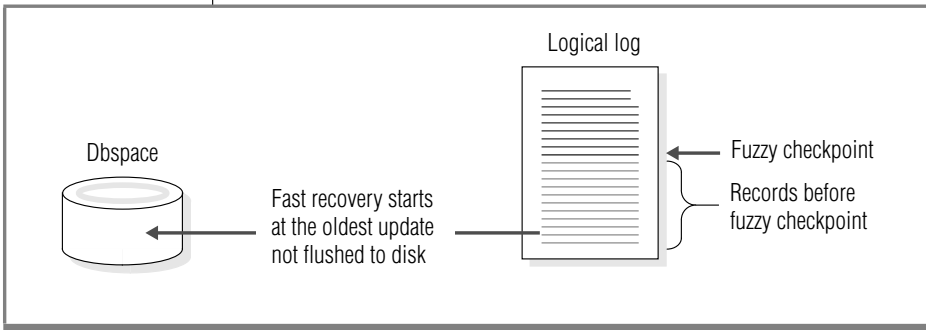
Figure 15-10
*Writing Nonfuzzy
Before-Images in
the Physical Log
Back to Disk*

Pages on which fuzzy operations occurred are not physically consistent because the database server does not physically log the before-images. If the most recent checkpoint was a fuzzy checkpoint, the changed pages for fuzzy operations were not flushed to disk. The dbspace disk still contains the before-image of each page. To undo changes to these pages prior to the fuzzy checkpoint, the database server uses the logical log, as the next step describes.

Locating the Oldest Update in the Logical Log

The database server no longer starts fast recovery at the most recent checkpoint record. In this step, the database server locates the oldest update record in the logical log that was not flushed to disk during the most recent checkpoint. The database server uses the log sequence numbers (LSN) in the logical log to find the oldest update record. Figure 15-11 shows that the oldest update in the logical log occurred several checkpoints ago and that all the log records are applied.

Figure 15-11
Locating the Oldest Update Record in the Logical Log



The LSN cannot be further back in the log than one logical-log file and two checkpoints. At each fuzzy checkpoint, the database server writes pages with time stamps earlier than the oldest LSN and moves forward the LSN.

You cannot free the logical log that contains the oldest update record until after the changes are recorded on disk. The database server automatically performs a full checkpoint to prevent problems with fast recovery of very old log records.

Applying the Log Records for Fuzzy Operations

In the second step, the database server processes the log records for fuzzy operations that occurred following the oldest update and before the last checkpoint. The log records that represent changes to data must be on disk before the changed data replaces the previous version on disk.

Log records for fuzzy operations are selectively redone, depending on whether the update has already been applied to the page. If the time stamp in the logical-log record is older than the time stamp in the disk page, the database server applies the record. Otherwise, the database server skips that record.

Figure 15-12 illustrates how the database server processes fuzzy records only prior to checkpoint.

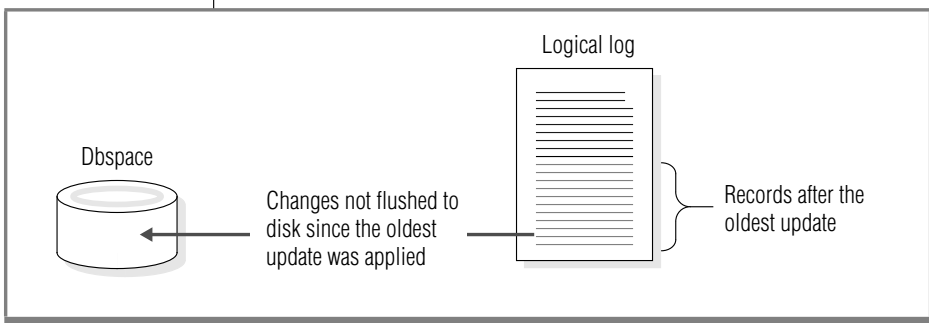


Figure 15-12
*Applying the Log
Records for Fuzzy
Operations*

Rolling Forward Logical-Log Records

In the third step, the database server processes all logical-log records following the last checkpoint. Fast recovery rolls forward the logical-log records that were written after the most recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point at which the uncontrolled shutdown occurred. Figure 15-13 illustrates the roll forward of all records after the fuzzy checkpoint.

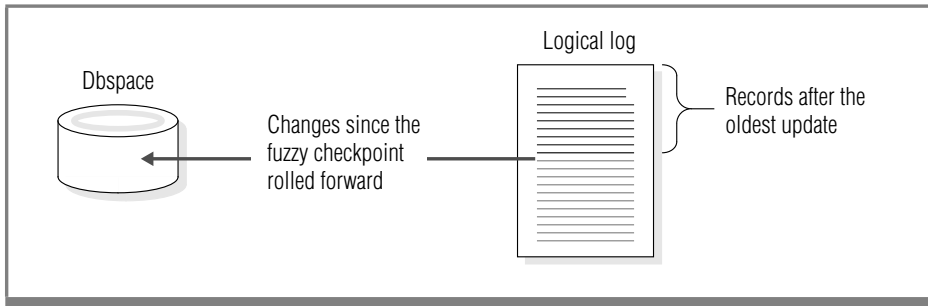


Figure 15-13
*Rolling Forward the
Logical-Log
Records Written
Since the Most
Recent Fuzzy
Checkpoint*

Rolling Back Incomplete Transactions

The final step in fast recovery rolls back all logical-log records for transactions that were not committed at the time that the system failed. This rollback procedure ensures that all databases are left in a consistent state.

Because one or more transactions possibly spanned several checkpoints without being committed, this rollback procedure might read backward through the logical log past the most recent checkpoint record. All logical-log files that contain records for open transactions are available to the database server because a log file is not freed until all transactions contained within it are closed. [Figure 15-14](#) illustrates the rollback procedure. When fast recovery is complete, the database server goes to quiescent or online mode.

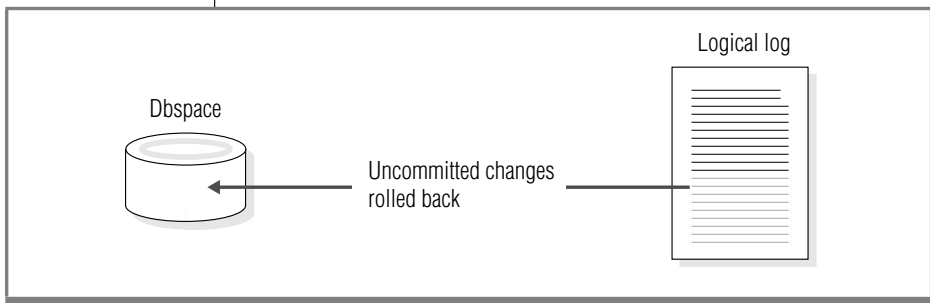


Figure 15-14
*Rolling Back All
Incomplete
Transactions*

Managing the Physical Log

In This Chapter	16-3
Changing the Physical-Log Location and Size	16-3
Preparing to Make the Changes	16-4
Checking For Adequate Contiguous Space	16-4
Using a Text Editor to Change Physical-Log Location or Size	16-5
Using onparams to Change Physical-Log Location or Size	16-5
Using ON-Monitor to Change Physical-Log Location or Size	16-6
Monitoring Physical and Logical-Logging Activity	16-6
Sample onstat -l Output	16-8
Monitoring Checkpoint Information	16-8
Forcing a Full Checkpoint	16-9
Forcing a Fuzzy Checkpoint	16-10
Using SMI Tables	16-11

In This Chapter

This chapter describes the following procedures:

- Changing the location and size of the physical log
- Monitoring the physical log, physical-log buffers, and logical-log buffers
- Monitoring and forcing checkpoints

For background information about the physical log, checkpoints, and fast recovery, see [Chapter 15, “Physical Logging, Checkpoints, and Fast Recovery.”](#)

Changing the Physical-Log Location and Size

You can change your physical-log location or size in several ways:

- Using a text editor or ISA to edit the ONCONFIG file
- Using the onparams utility from the command line
- Using ON-Monitor ♦

Log in as user informix or root on UNIX or as a member of the **Informix-Admin** group on Windows when you make the changes. The following sections describe each of these methods.

To activate the changes to the size or location of the physical log as soon as you make them, shut down and restart the database server. The **onparams** utility automatically shuts down and restarts the database server.

Create a level-0 backup immediately after you restart the database server. This storage-space backup is critical for database server recovery.

You can move the physical-log file to try to improve performance. When the database server initializes disk space, it places the disk pages allocated for the logical log and the physical log in the root dbspace. You might improve performance by moving the physical log, the logical-log files, or both to other dbspaces.

For advice on where to place the physical log, see [“Specifying the Location of the Physical Log” on page 15-6](#). For advice on sizing the physical log, see [“Size and Location of the Physical Log” on page 15-6](#). To obtain information about the physical log, see [“Monitoring Physical and Logical-Logging Activity” on page 16-6](#).

Preparing to Make the Changes

The space allocated for the physical log must be contiguous. A fatal shared-memory error occurs when you attempt to shut down and restart the database server in these situations:

- If you move the physical log to a dbspace without adequate contiguous space
- If you increase the log size beyond the available contiguous space, a fatal shared-memory error occurs when you attempt to restart the database server with the new values.

If this error occurs, resize the physical log, or choose another dbspace with adequate contiguous space and then shut down and restart the database server.

Checking For Adequate Contiguous Space

You can check whether adequate contiguous space is available with the **oncheck -pe** option. For information on using the **oncheck -ce** and **-pe** options to check the chunk free list, see the utilities chapter in the *Administrator's Reference*.

For more information, see [“Monitoring Chunks” on page 10-41](#).

Using a Text Editor to Change Physical-Log Location or Size

You can change the physical-log location and size by editing the ONCONFIG file while the database server is in online mode.

Parameter	Description
PHYSFILE	Specifies the size of the physical log file in kilobytes
PHYSDBS	Moves the physical log to the specified dbspace

The changes do not take effect until you shut down and restart the database server. Then create a level-0 backup immediately to ensure that all recovery mechanisms are available.

For information on PHYSFILE and PHYSDBS, see the chapter on configuration parameters in the *Administrator's Reference*.

Using onparams to Change Physical-Log Location or Size

To change the size and location of the physical log, execute the following command after you bring the database server to quiescent mode:

```
onparams -p -s size -d dbspace -y
```

size is the new size of the physical log in kilobytes

dbspace specifies the dbspace where the physical log is to reside

The following example changes the size and location of the physical log. The new physical-log size is 400 kilobytes, and the log will reside in the **dbspace6** dbspace. The command also reinitializes shared memory with the **-y** option so that the change takes effect immediately, as follows:

```
onparams -p -s 400 -d dbspace6 -y
```

After you shut down and restart the database server, create a level-0 backup to ensure that all recovery mechanisms are available.

For information on using the onparams utility to modify the physical log, see the utilities chapter in the *Administrator's Reference*.

UNIX

Using ON-Monitor to Change Physical-Log Location or Size

Select **Parameters**→**Physical-Log** to change the size or dbspace location, or both.

Then, create a level-0 backup immediately to ensure that all recovery mechanisms are available.

Monitoring Physical and Logical-Logging Activity

Monitor the physical log to determine the percentage of the physical-log file that gets used before a checkpoint occurs. This information allows you to find the optimal size of the physical-log file. It should be large enough that the database server does not have to force checkpoints too frequently and small enough to conserve disk space and guarantee fast recovery.

Monitor physical-log and logical-log buffers to determine if they are the optimal size for the current level of processing. The important statistic to monitor is the pages-per-disk-write statistic. For more information on tuning the physical-log and logical-log buffers, see your *Performance Guide*.

To monitor the physical-log file, physical-log buffers, and logical-log buffers, use the following commands.

Utility	Command	Additional Information
Command line or ISA	onstat -l	<p>The first line displays the following information for each physical-log buffer:</p> <ul style="list-style-type: none"> ■ The number of buffer pages used (bufused) ■ The size of each physical log buffer in pages (bufsize) ■ The number of pages written to the buffer (numpages) ■ The number of writes from the buffer to disk (numwrits) ■ The ratio of pages written to the buffer to the number of writes to disk (pages/IO) <p>The second line displays the following information about the physical log:</p> <ul style="list-style-type: none"> ■ The page number of the first page in the physical-log file (phybegin) ■ The size of the physical-log file in pages (physize) ■ The current position in the log where the next write occurs, specified as a page number (physpos) ■ The number of pages in the log that have been used (phyused) ■ The percentage of the total physical-log pages that have been used (%used) <p>The third line displays the following information about each logical-log buffer:</p> <ul style="list-style-type: none"> ■ The number of buffer pages used (bufused) ■ The size of each logical-log buffer in pages (bufsize) ■ The number of records written to the buffer (numrecs) ■ The number of pages written to the buffer (numpages) ■ The number of writes from the buffer to disk (numwrits) ■ The ratio of records to pages in the buffer (recs/pages) ■ The ratio of pages written to the buffer to the number of writes to disk (pages/IO)

(1 of 2)

Utility	Command	Additional Information
Command line or ISA	onparams -p	Moves or resizes the physical log.
Command line or ISA	onmode -l	Advances to the next logical-log file.
ISA	Logs→Logical	Click Advance Log File .

(2 of 2)

Sample onstat -l Output

Figure 16-1 shows sample output from the **onstat -l** option that shows information about the physical-log and logical-log buffers.

```
Physical Logging
Buffer bufused  bufsize  numpages numwrits pages/io
P-2  0          16      110       10       11.00
      phybegin physize  phypos   phyused  %used
10003f  500      233       0        0.00

Logical Logging
Buffer bufused  bufsize  numrecs  numpages numwrits  recs/pages pages/io
L-1  0          16      3075     162      75       19.0      2.2
...
```

Figure 16-1
onstat -l Output That Shows Information About the Physical and Logical Logs

Monitoring Checkpoint Information

Monitor checkpoint activity to determine basic checkpoint information. This information includes the number of times that threads had to wait for the checkpoint to complete. This information is useful for determining if the checkpoint interval is appropriate. For information on tuning the checkpoint interval, see your *Performance Guide*.

To monitor checkpoints, use the following commands.

Utility	Command	Additional Information
Command line or ISA	onstat -m	View the last 20 lines in the message log. If a checkpoint message does not appear in the last 20 lines, read the message log directly with a text editor. The database server writes individual checkpoint messages to the log when the checkpoint ends. If a checkpoint occurs, but the database server has no pages to write to disk, the database server does not write any messages to the message log.
Command line or ISA	onstat -p	Obtains these checkpoint statistics: numckpts Number of checkpoints that occurred since the database server was brought online. ckptwaits Number of times that a user thread waits for a checkpoint to finish. The database server prevents a user thread from entering a critical section during a checkpoint.
ON-Monitor (UNIX)	Status→Profile	The Checkpoints and Check Waits fields display the same information as the numckpts and ckpwaits fields in onstat -p .

Forcing a Full Checkpoint

To force a full checkpoint, execute one of the following commands.

Utility	Command	Additional Information
Command line or ISA	onmode -c	None.
ISA	Logs→Logical	Click Force Checkpoint .
ON-Monitor (UNIX)	Force-Ckpt option from the main menu	The time in the Last Checkpoint Done field does not change until a checkpoint occurs. The Last Checkpoint Check field shows the time of the last checkpoint check. If no modifications have been made since the time of the last checkpoint, the database server does not perform a checkpoint.

You should force a full checkpoint in any of the following situations:

- To free a logical-log file that contains the most recent checkpoint record and that is backed up but not yet released (onstat -l status of U-B-L or U-B)
- Before you issue **onmode -sy** to place the database server in quiescent mode
- After building a large index, if the database server terminates before the next checkpoint (the index build will restart the next time that you initialize the database server)
- If a checkpoint has not occurred for a long time and you are about to attempt a system operation that might interrupt the database server
- If foreground writes are taking more resources than you want (force a checkpoint to bring this down to zero for a while)
- Before you execute **dbexport** or unload a table, to ensure physical consistency of all data before you export or unload it
- After you perform a large load of tables using PUT or INSERT statements

Because table loads use the buffer cache, force a checkpoint to clean the buffer cache.

Forcing a Fuzzy Checkpoint

To force a fuzzy checkpoint, execute the **onmode -c fuzzy** command.

Using SMI Tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The **sysprofile** table also provides the same checkpoint statistics that are available from the **onstat -p** option. These rows contain the following statistics.

Row	Description
plgpagewrites	Number of pages written to the physical-log buffer
plgwrites	Number of writes from the physical-log buffer to the physical log file
llgreccs	Number of records written to the logical-log buffer
llgpagewrites	Number of pages written to the logical-log buffer
llgwrites	Number of writes from the logical-log buffer to the logical-log files
numckpts	Number of checkpoints that have occurred since the database server was brought online)
ckptwaits	Number of times that threads waited for a checkpoint to finish to enter a <i>critical section</i> during a checkpoint
value	Values for numckpts and ckptwaits

Fault Tolerance

- Chapter 17** **Mirroring**
- Chapter 18** **Using Mirroring**
- Chapter 19** **High-Availability Data Replication**
- Chapter 20** **Using High-Availability Data Replication**
- Chapter 21** **Consistency Checking**



Mirroring

In This Chapter	17-3
Mirroring	17-3
Benefits of Mirroring	17-4
Costs of Mirroring	17-4
Consequences of Not Mirroring	17-4
Data to Mirror	17-5
Alternatives to Mirroring	17-6
Logical Volume Managers	17-6
Hardware Mirroring	17-6
External Backup and Restore	17-7
Mirroring Process	17-7
Creation of a Mirrored Chunk	17-7
Mirror Status Flags	17-8
Recovery	17-9
Actions During Processing	17-9
Disk Writes to Mirrored Chunks	17-9
Disk Reads from Mirrored Chunks	17-10
Detection of Media Failures	17-10
Chunk Recovery	17-11
Result of Stopping Mirroring	17-11
Structure of a Mirrored Chunk	17-11

In This Chapter

This chapter describes the database server mirroring feature. For instructions on how to perform mirroring tasks, refer to [Chapter 18, “Using Mirroring.”](#)

Mirroring

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace, blobspace, or sbspace with an equal-sized *mirrored* chunk.

Every write to the primary chunk is automatically accompanied by an identical write to the mirrored chunk. This concept is illustrated in [Figure 17-1](#). If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirrored chunk until you can recover the primary chunk, all without interrupting user access to data.

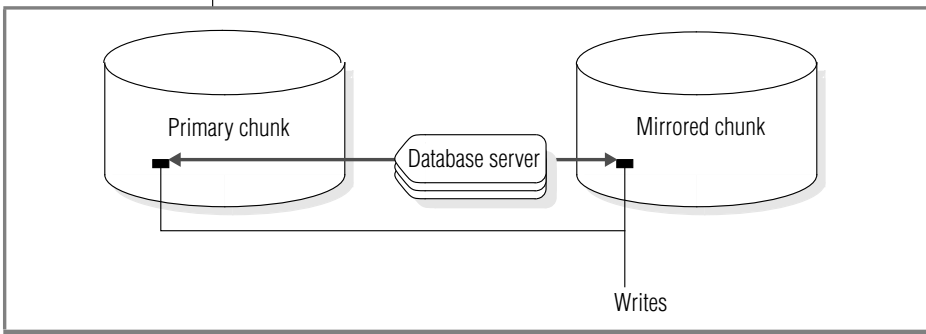


Figure 17-1
*Writing Data to Both
the Primary Chunk
and the Mirrored
Chunk*

Mirroring is not supported on disks that are managed over a network. The same database server instance must manage all the chunks of a mirrored set.

Benefits of Mirroring

If media failure occurs, mirroring provides the database server administrator with a means of recovering data without having to take the database server offline. This feature results in greater reliability and less system downtime. Furthermore, applications can continue to read from and write to a database whose primary chunks are on the affected media, provided that the chunks that mirror this data are located on separate media.

Any critical database should be located in a mirrored dbspace. Above all, the root dbspace, which contains the database server reserved pages, should be mirrored.

Costs of Mirroring

Disk-space costs as well as performance costs are associated with mirroring. The disk-space cost is due to the additional space required for storing the mirror data. The performance cost results from having to perform writes to both the primary and mirrored chunks. The use of multiple virtual processors for disk writes reduces this performance cost. The use of *split reads*, whereby the database server reads data from either the primary chunk or the mirrored chunk, depending on the location of the data within the chunk, actually causes performance to improve for read-only data. For more information on how the database server performs reads and writes for mirrored chunks, see [“Actions During Processing” on page 17-9](#).

Consequences of Not Mirroring

If you do not mirror your dbspaces, the frequency with which you have to restore from a storage-space backup after media failure increases.

When a mirrored chunk suffers media failure, the database server reads exclusively from the chunk that is still online until you bring the down chunk back online. On the other hand, when an *unmirrored* chunk goes down, the database server cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, the database server goes offline immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, the database server can continue to operate, but threads cannot read from or write to the down chunk. Unmirrored chunks that go down must be restored by recovering the dbspace from a backup.

Data to Mirror

Ideally, you should mirror all of your data. If disk space is an issue, however, you might not be able to do so. In this case, select certain critical chunks to mirror.

Critical chunks always include the chunks that are part of the root dbspace, the chunk that stores the logical-log files, and the chunk that stores the physical logs. If any one of these critical chunks fail, the database server goes offline immediately.

If some chunks hold data that is critical to your business, give these chunks high priority for mirroring.

Also give priority for mirroring to chunks that store frequently used data. This action ensures that the activities of many users would not be halted if one widely used chunk goes down.

Alternatives to Mirroring

Mirroring, as discussed in this manual, is a database server feature. Your operating system or hardware might provide alternative mirroring solutions.

If you are considering a mirroring feature provided by your operating system instead of database server mirroring, compare the implementation of both features before you decide which to use. The slowest step in the mirroring process is the actual writing of data to disk. The database server strategy of performing writes to mirrored chunks in parallel helps to reduce the time required for this step. (See [“Disk Writes to Mirrored Chunks” on page 17-9.](#)) In addition, database server mirroring uses split reads to improve read performance. (See [“Disk Reads from Mirrored Chunks” on page 17-10.](#)) Operating-system mirroring features that do not use parallel mirror writes and split reads might provide inferior performance.

Nothing prevents you from running database server mirroring and operating-system mirroring at the same time. They run independently of each other. In some cases, you might decide to use both the database server mirroring and the mirroring feature provided by your operating system. For example, you might have both database server data and other data on a single disk drive. You could use the operating-system mirroring to mirror the other data and database server mirroring to mirror the database server data.

Logical Volume Managers

Logical volume managers are an alternative mirroring solution. Some operating-system vendors provide this type of utility to have multiple disks appear as one file system. Saving data to more than two disks gives you added protection from media failure, but the additional writes have a performance cost.

Hardware Mirroring

Another solution is to use hardware mirroring such as RAID (redundant array of inexpensive disks). An advantage of this type of hardware mirroring is that it requires less disk space than database server mirroring does to store the same amount of data to prevent media failure.



Some hardware mirroring systems support *hot swapping*. You can swap a bad disk while keeping the database server online. Reducing I/O activity before performing a hot swap is recommended.

Important: *If problems occur with the database server while using hardware mirroring, refer to the operating-system or disk documentation or technical support for assistance.*

External Backup and Restore

If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional ON-Bar commands. For more information on external backup and restore, see the *IBM Informix Backup and Restore Guide*.

Mirroring Process

This section describes the mirroring process in greater detail. For instructions on how to perform mirroring operations such as creating mirrored chunks, starting mirroring, changing the status of mirrored chunks, and so forth, see [Chapter 18, “Using Mirroring.”](#)

Creation of a Mirrored Chunk

When you specify a mirrored chunk, the database server copies all the data from the primary chunk to the mirrored chunk. This copy process is known as *recovery*. Mirroring begins as soon as recovery is complete.

The recovery procedure that marks the beginning of mirroring is delayed if you start to mirror chunks within a dbspace that contains a logical-log file. Mirroring for dbspaces that contain a logical-log file does not begin until you create a level-0 backup of the root dbspace. The delay ensures that the database server can use the mirrored logical-log files if the primary chunk that contains these logical-log files becomes unavailable during a dbspace restore.

The level-0 backup copies the updated database server configuration information, including information about the new mirrored chunk, from the root dbspace reserved pages to the backup. If you perform a data restore, the updated configuration information at the beginning of the backup directs the database server to look for the mirrored copies of the logical-log files if the primary chunk becomes unavailable. If this new storage-space backup information does not exist, the database server is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbspace that contains a logical-log file while a dbspace backup is being created. The new information that must appear in the first block of the dbspace backup tape cannot be copied there once the backup has begun.

For more information on creating mirrored chunks, refer to [Chapter 18, “Using Mirroring.”](#)

Mirror Status Flags

Dbspaces, blobspaces, and sbspaces have status flags that indicate whether they are mirrored or unmirrored.

You must perform a level-0 backup of the root dbspace before mirroring starts.

Chunks have status flags that indicate the following information:

- Whether the chunk is a primary or mirrored chunk
- Whether the chunk is currently online, down, a new mirrored chunk that requires a level-0 backup of the root dbspace, or in the process of being recovered

For descriptions of these chunk status flags, refer to the description of the **onstat -d** option in the utilities chapter of the *Administrator's Reference*. For information on how to display these status flags, refer to [“Monitoring Disk Usage” on page 10-41](#).

Recovery

When the database server recovers a mirrored chunk, it performs the same recovery procedure that it uses when mirroring begins. The mirror-recovery process consists of copying the data from the existing online chunk onto the new, repaired chunk until the two are identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the online chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives online status. You perform the same steps whether you are recovering the primary chunk of a mirrored pair or recovering the mirrored chunk.



Tip: *You can still use the online chunk during the recovery process. If data is written to a page that has already been copied to the recovery chunk, the database server updates the corresponding page on the recovery chunk before it continues with the recovery process.*

For information on how to recover a down chunk, refer to the information on recovering a mirrored chunk on [page 18-11](#).

Actions During Processing

This section discusses some of the details of disk I/O for mirrored chunks and how the database server handles media failure for these chunks.

Disk Writes to Mirrored Chunks

During database server processing, the database server performs mirroring by executing two parallel writes for each modification: one to the primary chunk and one to the mirrored chunk.

Disk Reads from Mirrored Chunks

The database server uses mirroring to improve read performance because two versions of the data reside on separate disks. A data page is read from either the primary chunk or the mirrored chunk, depending on which half of the chunk includes the address of the data page. This feature is called a *split read*. Split reads improve performance by reducing the disk-seek time. Disk-seek time is reduced because the maximum distance over which the disk head must travel is reduced by half. [Figure 17-2](#) illustrates a split read.

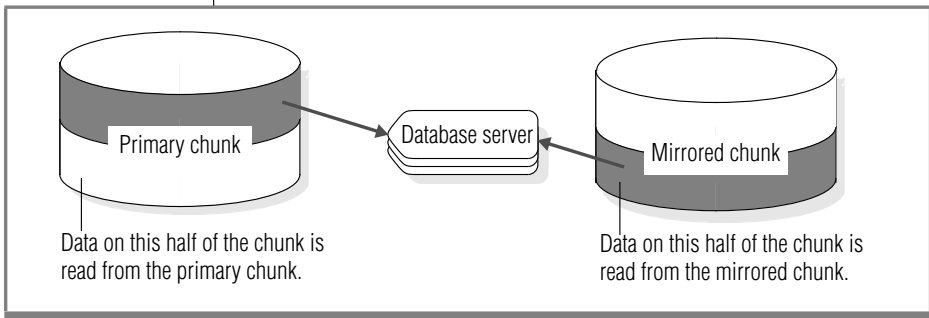


Figure 17-2
*Split Read Reducing
the Maximum
Distance Over
Which the Disk Head
Must Travel*

Detection of Media Failures

The database server checks the return code when it first opens a chunk and after any read or write. Whenever the database server detects that a primary (or mirror) chunk device has failed, it sets the chunk-status flag to down (D). For information on chunk-status flags, refer to [“Mirror Status Flags” on page 17-8](#).

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains online. This statement is true even if the administrator intentionally brings down one of the chunks.

Once the administrator recovers the down chunk and returns it to online status, reads are again split between the primary and mirrored chunks, and writes are made to both chunks.

Chunk Recovery

The database server uses asynchronous I/O to minimize the time required for recovering a chunk. The read from the chunk that is online can overlap with the write to the down chunk, instead of the two processes occurring serially. That is, the thread that performs the read does not have to wait until the thread that performs the write has finished before it reads more data.

Result of Stopping Mirroring

When you end mirroring, the database server immediately frees the mirrored chunks and makes the space available for reallocation. The action of ending mirroring takes only a few seconds.

Create a level-0 backup of the root dbspace after you end mirroring to ensure that the reserved pages with the updated mirror-chunk information are copied to the backup. This action prevents the restore procedure from assuming that mirrored data is still available.

Structure of a Mirrored Chunk

The mirrored chunk contains the same control structures as the primary chunk, as follows:

- Mirrors of blob space chunks contain blob space overhead pages.
- Mirrors of db space chunks contain db space overhead pages.
- Mirrors of sb spaces contain metadata pages.

For information on these structures, refer to the section on the structure of a mirrored chunk in the disk structures and storage chapter of the *Administrator's Reference*.

A display of disk-space use, provided by one of the methods discussed under [“Monitoring Chunks” on page 10-41](#), always indicates that the mirrored chunk is full, even if the primary chunk has free space. The *full* mirrored chunk indicates that none of the space in the chunk is available for use other than as a mirror of the primary chunk. The status remains full for as long as both primary chunk and mirrored chunk are online.

If the primary chunk goes down and the mirrored chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Using Mirroring

In This Chapter	18-3
Preparing to Mirror Data	18-3
Enabling the MIRROR Configuration Parameter.	18-4
Changing the MIRROR Parameter with ON-Monitor.	18-4
Allocating Disk Space for Mirrored Data	18-5
Linking Chunks	18-5
Relinking a Chunk to a Device After a Disk Failure	18-5
Using Mirroring	18-6
Mirroring the Root Dbspace During Initialization	18-7
Changing the Mirror Status	18-7
Managing Mirroring	18-8
Starting Mirroring for Unmirrored Storage Spaces.	18-8
Starting Mirroring for New Storage Spaces	18-9
Adding Mirrored Chunks	18-10
Taking Down a Mirrored Chunk	18-10
Recovering a Mirrored Chunk.	18-11
Ending Mirroring	18-11

In This Chapter

This chapter describes the various mirroring tasks that are required to use the database server mirroring feature. It provides an overview of the steps required for mirroring data.

Preparing to Mirror Data

This section describes how to start mirroring data on a database server that is not running with the mirroring function enabled.

To prepare to mirror data

1. Take the database server offline and enable mirroring.
See [“Enabling the MIRROR Configuration Parameter” on page 18-4](#).
2. Bring the database server back online.
3. Allocate disk space for the mirrored chunks.

You can allocate this disk space at any time, as long as the disk space is available when you specify mirrored chunks in the next step. The mirrored chunks should be on a different disk than the corresponding primary chunks. See [“Allocating Disk Space for Mirrored Data” on page 18-5](#).

4. Choose the dbspace, blobspace, or sbospace that you want to mirror, and specify a mirror-chunk pathname and offset for each primary chunk in that storage space.

The mirroring process starts after you perform this step. Repeat this step for all the storage spaces that you want to mirror. See [“Using Mirroring” on page 18-6](#).

Enabling the MIRROR Configuration Parameter

Enabling mirroring invokes the database server functionality required for mirroring tasks. However, when you enable mirroring, you do not initiate the mirroring process. Mirroring does not actually start until you create mirrored chunks for a dbspace, blob space, or sbspace. See [“Using Mirroring” on page 18-6](#).

Enable mirroring when you initialize the database server if you plan to create a mirror for the root dbspace as part of initialization; otherwise, leave mirroring disabled. If you later decide to mirror a storage space, you can change the value of the MIRROR configuration parameter.

To enable mirroring for the database server, you must set the MIRROR parameter in **ONCONFIG** to 1. The default value of MIRROR is 0, indicating that mirroring is disabled.

To change the value of MIRROR, you can edit the **ONCONFIG** file with a text editor or ISA while the database server is in online mode. After you change the **ONCONFIG** file, take the database server offline and then to quiescent mode for the change to take effect.

UNIX

Changing the MIRROR Parameter with ON-Monitor

To enable mirroring, choose **Parameters→Initialize**. In the field that is labelled **Mirror**, enter **Y**. Press **ESC** to record changes.

After the last of these screens, a prompt appears to confirm that you want to continue (to initialize the database server disk space and destroy all existing data). Respond **N** (no) to this prompt.

Warning: *If you respond Y (yes) at this prompt, you lose all your existing data.*

Take the database server offline and then to quiescent mode for the change to take effect.



UNIX

Allocating Disk Space for Mirrored Data

Before you can create a mirrored chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirrored chunks. For a discussion of allocating disk space, refer to [“Allocating Disk Space” on page 10-6](#).

Always allocate disk space for a mirrored chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. This setup allows you to access the mirrored chunk if the disk on which the primary chunk is located goes down, or vice versa.

Linking Chunks

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirrored chunks to mirror pathnames. If a disk failure occurs, you can link a new file or raw device to the pathname, eliminating the need to physically replace the disk that failed before the chunk is brought back online.

Relinking a Chunk to a Device After a Disk Failure

On UNIX, if the disk on which the actual mirror file or raw device is located goes down, you can relink the chunk to a file or raw device on a different disk. This action allows you to recover the mirrored chunk before the disk that failed is brought back online. Typical UNIX commands that you can use for relinking are shown in the following examples.

The original setup consists of a primary root chunk and a mirror root chunk, which are linked to the actual raw disk devices, as follows:

```
ln -l
lrwxrwxrwx 1 informix 10 May 3 13:38 /dev/root@->/dev/rxy0h
lrwxrwxrwx 1 informix 10 May 3 13:40 /dev/mirror_root@->/dev/rsd2b
```

Assume that the disk on which the raw device **/dev/rsd2b** resides has gone down. You can use the **rm** command to remove the corresponding symbolic link, as follows:

```
rm /dev/mirror_root
```

Now you can relink the mirrored chunk pathname to a raw disk device, on a disk that is running, and proceed to recover the chunk, as follows:

```
ln -s /dev/rab0a /dev/mirror_root
```

Using Mirroring

Mirroring starts when you create a mirrored chunk for each primary chunk in a dbspace, blobspace, or sbspace.

When you create a mirrored chunk, the database server copies data from the primary chunk to the mirrored chunk. When this process is complete, the database server begins mirroring data. If the primary chunk contains logical-log files, the database server does not copy the data immediately after you create the mirrored chunk but waits until you perform a level-0 backup. For an explanation of this behavior see [“Creation of a Mirrored Chunk” on page 17-7](#).



Important: You must always start mirroring for an entire dbspace, blobspace, or sbspace. The database server does not permit you to select particular chunks in a dbspace, blobspace, or sbspace to mirror. You must create mirrored chunks for every chunk in the space.

You start mirroring a storage space when you perform the following operations:

- Create a mirrored root dbspace during system initialization
- Change the status of a dbspace from unmirrored to mirrored
- Create a mirrored dbspace, blobspace, or sbspace

Each of these operations requires you to create mirrored chunks for the existing chunks in the storage space.

Mirroring the Root Dbspace During Initialization

If you enable mirroring when you initialize the database server, you can also specify a mirror pathname and offset for the root chunk. The database server creates the mirrored chunk when the server is initialized. However, because the root chunk contains logical-log files, mirroring does not actually start until you perform a level-0 backup.

To specify the root mirror pathname and offset, set the values of `MIRRORPATH` and `MIRROROFFSET` in the `ONCONFIG` file before you bring up the database server.

If you do not provide a mirror pathname and offset, but you do want to start mirroring the root dbspace, you must change the mirroring status of the root dbspace once the database server is initialized.

Changing the Mirror Status

You can make the following two changes to the status of a mirrored chunk:

- Change a mirrored chunk from online to down
- Change a mirrored chunk from down to recovery

You can take down or restore a chunk only if it is part of a mirrored pair. You can take down either the primary chunk or the mirrored chunk, as long as the other chunk in the pair is online.

For information on how to determine the status of a chunk, refer to [“Monitoring Disk Usage” on page 10-41](#).

Managing Mirroring

You can use the **onspaces** utility to manage mirroring. On UNIX, you can also use ON-Monitor to manage mirroring. For a full description of the **onspaces** syntax, see the utilities chapter of the *Administrator's Reference*.

Starting Mirroring for Unmirrored Storage Spaces

You can prepare mirroring for a dbspace, blobspace, or sbspace at any time. However, the mirroring does not start until you perform a level-0 backup.

To start mirroring for unmirrored dbspaces with onspaces

You can use the **onspaces** utility to start mirroring a dbspace, blobspace, or sbspace. For example, the following **onspaces** command starts mirroring for the dbspace **db_project**, which contains two chunks, **data1** and **data2**:

```
onspaces -m db_project\  
-p /dev/data1 -o 0 -m /dev/mirror_data1 0\  
-p /dev/data2 -o 5000 -m /dev/mirror_data2 5000
```

The following example shows how to turn on mirroring for a dbspace called **sp1**. You can either specify the primary path, primary offset, mirror path, and mirror offset in the command or in a file.

```
onspaces -m sp1 -f mirfile
```

The **mirfile** file contains the following line:

```
/ix/9.3/sp1 0 /ix/9.2/sp1mir 0
```

In this line, **/ix/9.3/sp1** is the primary path, 0 is the primary offset, **/ix/9.3/sp1mir** is the mirror path, and 0 is the mirror offset.

To start mirroring with ISA

1. Select **Storage→Chunks**.
2. Select the dbspace name and click **Start Mirroring**.

UNIX

To start mirroring for unmirrored dbspaces with ON-Monitor

Use the **Dbspaces→Mirror** option to start mirroring a dbspace. To select the dbspace that you want to mirror, move the cursor down the list to the correct dbspace and press CTRL-B. The **Mirror** option then displays a screen for each chunk in the dbspace. You can enter a **mirror** pathname and offset in this screen. After you enter information for each chunk, press ESC to exit the option. The database server recovers the new mirrored chunks, meaning it copies data from the primary to the mirror chunk. If the chunk contains logical-log files, recovery is postponed until after you create a level-0 backup.

Starting Mirroring for New Storage Spaces

You can also start mirroring when you create a new dbspace, blobspace, or sbspace.

To start mirroring for new storage spaces with onspaces

You can use the **onspaces** utility to create a mirrored dbspace. For example, the following command creates the dbspace **db_acct** with an initial chunk **/dev/chunk1** and a mirrored chunk **/dev/mirror_chk1**:

```
onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

To start mirroring for new storage spaces with ISA

1. Select **Storage→Spaces**.
2. Click **Add Dbspace**, **Add Blobspace**, or **Add Sbspace**.
3. Enter the path and offset for the mirror chunk.

Another way to start mirroring is to select **Index by Utility→onspaces→-m**.

UNIX

To start mirroring for new dbspaces with ON-Monitor

To create a dbspace with mirroring, choose the **Dbspaces→Create** option. This option displays a screen in which you can specify the pathname, offset, and size of a primary chunk and the pathname and offset of a mirrored chunk for the new dbspace.

Adding Mirrored Chunks

If you add a chunk to a dbspace, blobspace, or sbpace that is mirrored, you must also add a corresponding mirrored chunk.

To add mirrored chunks with onspaces

You can use the **onspaces** utility to add a primary chunk and its mirrored chunk to a dbspace, blobspace, or sbpace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Because the dbspace is mirrored, a mirrored chunk, **mirror_chk2**, is also added.

```
onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```

To add mirrored chunks with ISA

1. Select **Storage→Chunks**.
2. Select the dbspace name and click **Add Chunk**.
3. Enter the path and offset for the mirror chunk.

To add mirrored chunks with ON-Monitor

In ON-Monitor, the **Dbspaces→Add-chunk** option displays fields in which to enter the primary-chunk pathname, offset, and size, and the mirror-chunk pathname and offset.

Taking Down a Mirrored Chunk

When a mirrored chunk is *down*, the database server cannot write to it or read from it. You might take down a mirrored chunk to relink the chunk to a different device. (See [“Relinking a Chunk to a Device After a Disk Failure” on page 18-5](#).)

Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbspace, which causes the database server to drop all the mirrored chunks for that dbspace.

To take down a mirrored chunk with onspaces

You can use the **onspaces** utility to take down a chunk. The following example takes down a chunk that is part of the dbspace **db_acct**:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -D
```


UNIX

To take down a mirrored chunk with ON-Monitor

To use ON-Monitor to take down a mirrored chunk, choose the **Dbspaces→Status** option. With the cursor on the dbspace that contains the chunk that you want to take down, press F3 or CTRL-B. The database server displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that you want to take down and press F3 or CTRL-B to change the status (take it down).

Recovering a Mirrored Chunk

To begin mirroring the data in the chunk that is online, you must recover the down chunk.

To recover a mirrored chunk with onspaces

You can use the **onspaces -s** utility to recover a down chunk. For example, to recover a chunk that has the pathname **/dev/mirror_chk1** and an offset of 0 kilobytes, issue the following command:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```

To recover a mirrored chunk with ISA

Select **Index by Utility→onspaces→-s**.

UNIX

To recover a mirrored chunk with ON-Monitor

To use ON-Monitor to recover a down chunk, choose the **Dbspaces→Status** option.

Ending Mirroring

When you end mirroring for a dbspace, blobspace, or sbpace, the database server immediately releases the mirrored chunks of that space. These chunks are immediately available for reassignment to other storage spaces.

Only users **informix** and **root** on UNIX or members of the **Informix-Admin** group on Windows can end mirroring.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in online mode when you end mirroring.

UNIX

To end mirroring with onspaces

You can end mirroring with the **onspaces** utility. For example, to end mirroring for the root dbspace, enter the following command:

```
onspaces -r rootdbs
```

To end mirroring with ON-Monitor

To end mirroring for a dbspace or blobospace with ON-Monitor, select the **Dbspaces→Mirror** option. Select a dbspace or blobospace that is mirrored and press CTRL-B or F3.

To end mirroring with ISA

1. Select **Storage→Chunks**.
2. Select the dbspace name and click **Stop Mirroring**.

Another way to end mirroring is to select **Index by Utility→onspaces→-r**.

High-Availability Data Replication

In This Chapter	19-3
High-Availability Data Replication	19-4
Type of Data Replicated	19-4
Advantages of Data Replication	19-5
Primary and Secondary Database Servers	19-5
HDR Versus Mirroring	19-7
HDR Versus Two-Phase Commit	19-8
HDR and Enterprise Replication	19-9
How HDR Works	19-9
How Data Initially Replicates	19-9
Reproduction of Updates to the Primary Database Server	19-10
How the Log Records Are Sent	19-11
HDR Buffers	19-11
When Log Records Are Sent	19-12
Synchronous Updating	19-12
Asynchronous Updating	19-12
Threads That Handle HDR	19-15
Checkpoints Between Database Servers	19-16
How Data Synchronization Is Tracked	19-16
HDR Failures	19-17
HDR Failures Defined	19-17
Detection of HDR Failures	19-18
Actions When an HDR Failure Is Detected	19-18
Considerations After HDR Failure	19-19
Actions to Take If the Secondary Database Server Fails	19-19
Actions to Take If the Primary Database Server Fails	19-20

Redirection and Connectivity for Data-Replication Clients	19-21
Designing Clients for Redirection	19-21
Directing Clients Automatically with DBPATH	19-22
How the DBPATH Redirection Method Works	19-22
What the Administrator Needs to Do	19-23
What the User Needs to Do	19-23
Directing Clients with the Connectivity Information	19-23
How Redirection with the Connectivity Information Works	19-24
Changing the Connectivity Information	19-24
Connecting to the Database Server	19-27
Directing Clients with INFORMIXSERVER	19-27
How Redirection Works with INFORMIXSERVER	19-27
What the Administrator Needs to Do	19-28
What the User Needs to Do	19-28
Handling Redirection Within an Application.	19-28
Comparing Different Redirection Mechanisms	19-31
Designing HDR Clients	19-31
Setting Lock Mode to Wait for Access to Primary Database Server	19-32
Designing Clients to Use the Secondary Database Server	19-33
No Data Modification Statements	19-33
Locking and Isolation Level	19-33
Use of Temporary Dbspaces for Sorting and Temporary Tables	19-35

In This Chapter

Data replication refers to the process of representing database objects at more than one distinct site. *High-Availability Data Replication* (HDR) provides synchronous data replication for Dynamic Server. Use HDR if you require a hot standby.

The chapter discusses the following topics:

- What HDR is
- How HDR works
- How HDR handles failures
- How to redirect a client to connect to the other database server in the HDR pair (also called a *replication pair*)
- What the design considerations are for applications that connect to the secondary database server

Chapter 20, “Using High-Availability Data Replication,” contains instructions on how to accomplish the administrative tasks that are involved in using HDR.

Tip: *If you want to use asynchronous data replication, see the “IBM Informix Dynamic Server Enterprise Replication Guide.”*



High-Availability Data Replication

One way of replicating data is simply to copy a database to a database server installed on a different computer. This copy allows reports to access the data without disturbing client applications that use the original database.

The database server implements nearly transparent data replication of entire database servers. All the data that one database server manages is replicated and dynamically updated on another database server, often at a separate geographical location. HDR provides a way to maintain a backup copy of the entire database server that applications can access quickly in the event of a catastrophic failure.



Important: HDR does not support network encryption during replication.

Type of Data Replicated

HDR replicates data in dbspaces and sbspaces. HDR does not replicate data in blobspaces.

HDR replicates all built-in and extended data types. User-defined types (UDTs) must be logged and reside in a single database server. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server. For data stored in an sbspace to be replicated, the sbspace must be logged.

HDR does not replicate data stored in operating system files or persistent external files or memory objects associated with user-defined routines.

User-defined types, user-defined routines, and DataBlade modules have special installation and registration requirements. For instructions, see [“How Data Initially Replicates” on page 19-9](#).

Advantages of Data Replication

Advantages of data replication are as follows:

- Clients at the site to which the data is replicated experience improved performance because those clients can access data locally rather than connecting to a remote database server over a network.
- Clients at all sites experience improved availability of replicated data. If the local copy of the replicated data is unavailable, clients can still access the remote copy of the data.

These advantages do not come without a cost. Data replication obviously requires more storage, and updating replicated data can take more processing time than updating a single object.

You could implement data replication in the logic of client applications by explicitly specifying where data must be updated. However, this method of achieving data replication is costly, prone to error, and difficult to maintain. Instead, the concept of data replication is often coupled with *replication transparency*. Replication transparency is built into a database server (instead of into client applications) to handle automatically the details of locating and maintaining data replicas.

Primary and Secondary Database Servers

When you configure a pair of database servers to use HDR, one database server is called the *primary* database server, and the other is called the *secondary* database server. (In this context, a database server that does not use HDR is referred to as a *standard* database server.)

During normal operation, clients can connect to the primary database server and use it as they would an ordinary database server. Clients can also use the secondary database server during normal operation, but only to read data. The secondary database server does not permit updates from client applications.

As [Figure 19-1](#) illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.

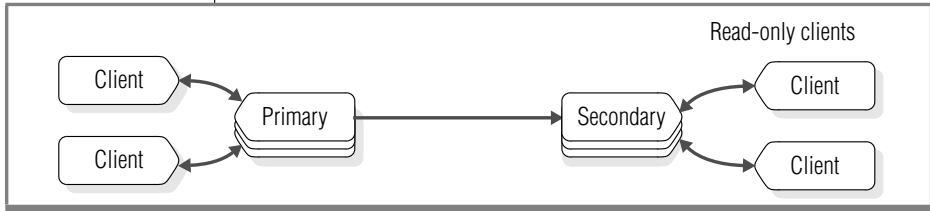


Figure 19-1
A Primary and Secondary Database Server in an HDR Pair

If one of the database servers fails, as [Figure 19-2](#) shows, you can redirect the clients that use that database server to the other database server in the pair.

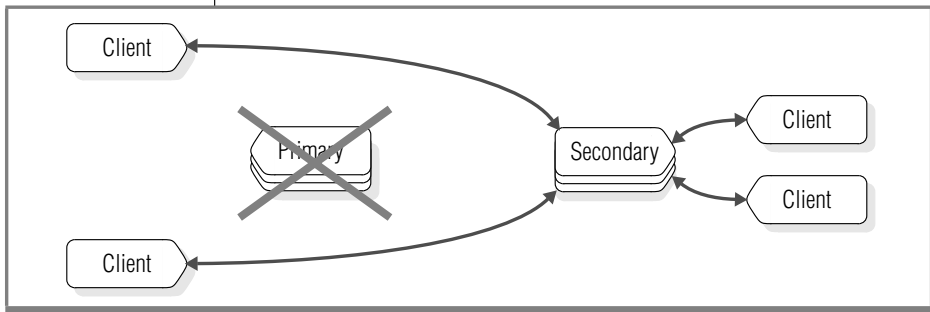


Figure 19-2
Database Servers in an HDR Pair and Clients After a Failure

If a primary database server fails, you can change the secondary database server to a standard database server so that it can accept updates.

HDR has the following features:

- Provides for quick recovery if one database server experiences a failure
- Allows for load balancing across the two database servers

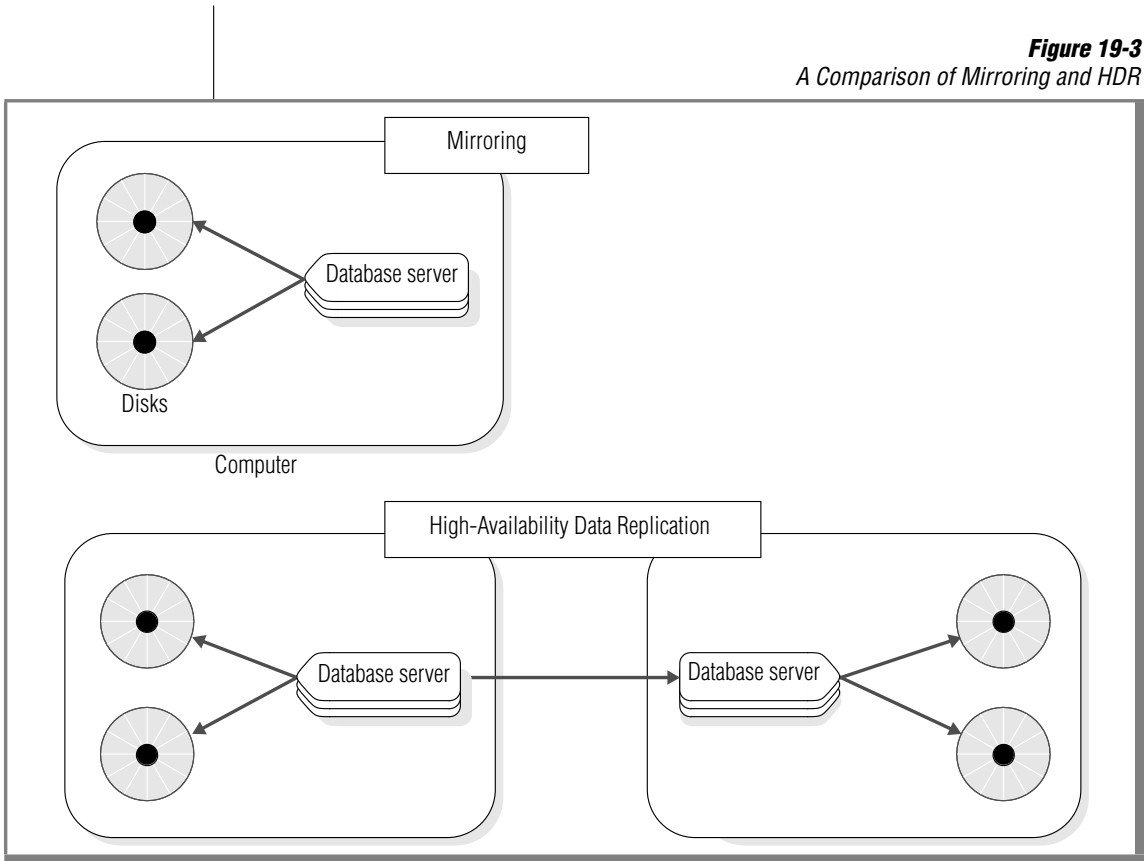
HDR Versus Mirroring

HDR and mirroring are both transparent ways of making the database server more fault tolerant. However, as [Figure 19-3](#), shows, they are quite different.

Mirroring, described in [“Mirroring” on page 17-3](#), is the mechanism by which a single database server maintains a copy of a specific dbspace on a separate disk. This mechanism protects the data in mirrored dbspaces against disk failure because the database server automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

On the other hand, HDR duplicates on an entirely separate database server all the data that a database server manages (not just specified dbspaces). Because HDR involves two separate database servers, it protects the data that these database servers manage, not just against disk failures, but against all types of database server failures, including a computer failure or the catastrophic failure of an entire site.

Figure 19-3
A Comparison of Mirroring and HDR



HDR Versus Two-Phase Commit

The two-phase commit protocol, described in detail in [Chapter 22, “Multiphase Commit Protocols,”](#) ensures that transactions are uniformly committed or rolled back across multiple database servers.

In theory, you could take advantage of two-phase commit to replicate data by configuring two database servers with identical data and then defining triggers on one of the database servers that replicate updates to the other database server. However, this sort of implementation has numerous synchronization problems in different failure scenarios. Also, the performance of distributed transactions is inferior to HDR.

HDR and Enterprise Replication

You can combine HDR with Enterprise Replication to create a robust replication system. HDR can ensure that an Enterprise Replication system remains fully connected by providing backup database servers for critical replication nodes.

When you combine HDR and Enterprise Replication, only the primary HDR server is connected to the Enterprise Replication system. The secondary HDR server does not participate in Enterprise Replication unless the primary HDR server fails.

For more information, see *IBM Informix Dynamic Server Enterprise Replication Guide*.

How HDR Works

This section describes the mechanisms that the database server uses to perform HDR. For instructions on how to set up, start, and administer an HDR system, refer to [Chapter 20, “Using High-Availability Data Replication,”](#) and initializing an HDR pair using external backup and restore in the *IBM Informix Backup and Restore Guide*.

How Data Initially Replicates

The database server uses storage-space and logical-log backups (both those backed up to tape and those on disk) to perform an initial replication of the data on the primary database server to the secondary database server.

To replicate data

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on the primary database server only.
2. To make the bulk of the data managed by the two database servers the same, create a level-0 backup of all the storage spaces on the primary database server, and restore all the storage spaces from that backup on the secondary database server in the data-replication pair.

The secondary database server that you restored from a storage-space backup in the previous step then reads all the logical-log records generated since that backup from the primary database server.

The database server reads the logical-log records first from any backed-up logical-log files that are no longer on disk and then from any logical-log files on disk.

For detailed instructions on performing the preceding steps, refer to [“Starting HDR for the First Time” on page 20-9](#). The *IBM Informix Backup and Restore Guide* discusses how to initialize replication using ON-Bar.

You must perform the initial HDR with a storage-space backup. You cannot use data-migration utilities such as **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical in order for HDR to work.

When HDR is working, the primary database server is online and accepts updates and queries just as a standard database server does. The secondary database server is in logical-recovery mode and cannot accept SQL statements that result in writes to disk (except for sorting and temporary tables).

Reproduction of Updates to the Primary Database Server

HDR reproduces updates to the primary database server on the secondary database server by having the primary database server send all its logical-log records to the secondary database server as they are generated. The secondary database server receives the logical-log records generated on the primary database server and applies them to its dbspaces.

Important: The database server cannot replicate updates to databases that do not use transaction logging.



How the Log Records Are Sent

As shown in [Figure 19-4 on page 19-11](#), when the primary database server starts to flush the contents of the logical-log buffer in shared memory to the logical log on disk, the database server also copies the contents of the logical-log buffer to a *data-replication buffer* on the primary database server. The primary database server then sends these logical-log records to the secondary database server.

The secondary database server receives the logical-log records from the primary database server into a shared-memory *reception buffer* (which the database server automatically adjusts to an appropriate size for the amount of data being sent). The secondary database server then applies the logical-log records through logical recovery.

HDR Buffers

The HDR buffers are part of the virtual shared memory that the primary database server manages. The HDR buffers hold logical-log records before the primary database server sends them to the secondary database server. The HDR buffers are the same size as the logical-log buffers. [Figure 19-4](#) shows this concept.

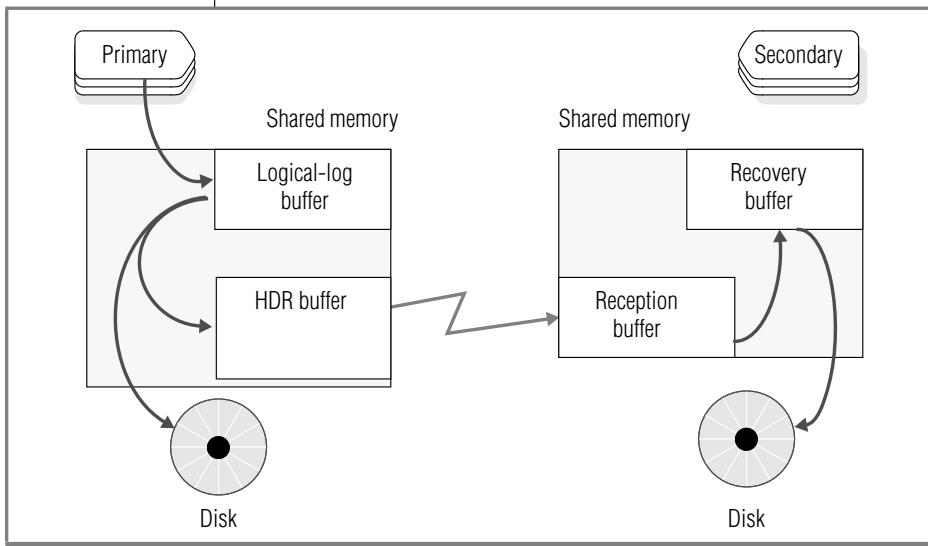


Figure 19-4
How Logical-Log Records Are Sent from the Primary Database Server to the Secondary Database Server

When Log Records Are Sent

The primary database server sends the contents of the HDR buffer to the secondary database server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL determines whether the database server uses synchronous or asynchronous updating. For more information on DRINTERVAL, see the chapter on configuration parameters in the *Administrator's Reference*.

Synchronous Updating

If you set DRINTERVAL to -1, HDR occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the HDR buffer, it sends those records from the HDR buffer to the secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the secondary database server that the records were received.

With synchronous updating, no transactions committed on the primary database server are left uncommitted or partially committed on the secondary database server if a failure occurs.

Asynchronous Updating

If you set DRINTERVAL to any value other than -1, data replication occurs *asynchronously*. The primary database server flushes the logical-log buffer after it copies the logical-log buffer contents to the HDR buffer. Independent of that action, the primary database server sends the contents of the HDR buffer across the network when one of the following conditions occurs:

- The HDR buffer becomes full.
- An application commits a transaction on an unbuffered database.
- The time interval, specified by the ONCONFIG parameter DRINTERVAL on the primary database server, has elapsed since the last time that records were sent to the secondary database server.

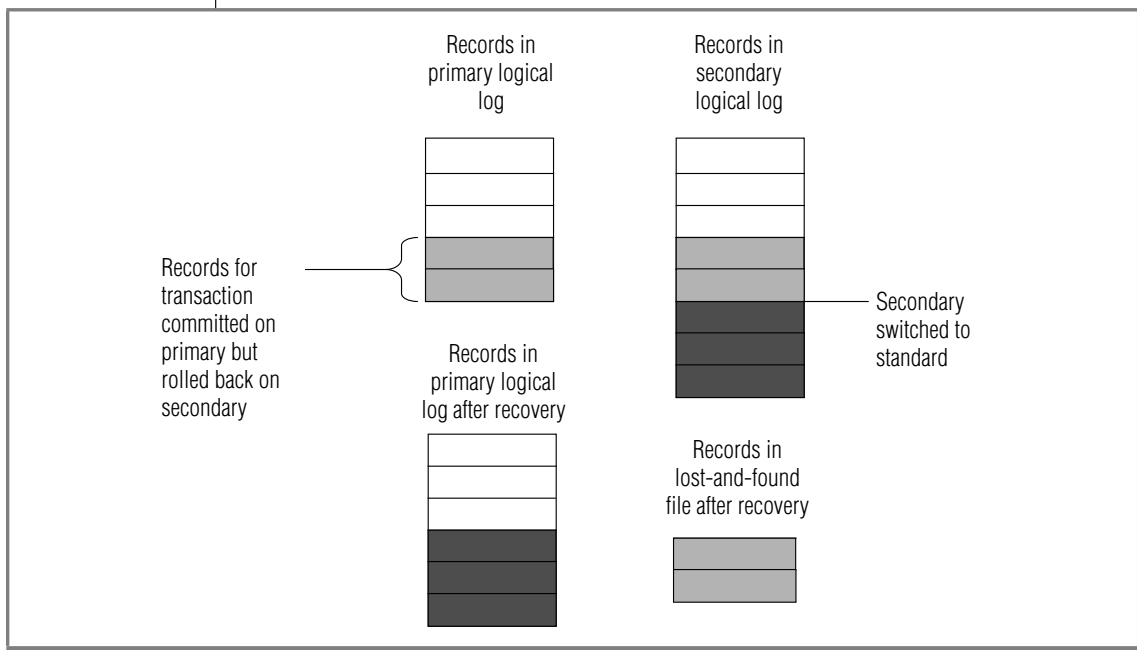
This method of updating might provide better performance than synchronous updating. However, as the following section explains, transactions may be lost.

Lost-and-Found Transactions

With asynchronous updating, a transaction committed on the primary database server might not be replicated on the secondary database server. This situation can result if a failure occurs after the primary database server copies a commit record to the HDR buffer but before the primary database server sends that commit record to the secondary database server.

If the secondary database server is changed to a standard database server after a failure of the primary database server, it rolls back any open transactions. These transactions include any that were committed on the primary database server but for which the secondary database server did not receive a commit record. As a result, transactions are committed on the primary database server but not on the secondary database server. When you restart data replication after the failure, the database server places all the logical-log records from the lost transactions in a file (which the ONCONFIG parameter DRLOSTFOUND specifies) during logical recovery of the primary database server. [Figure 19-5](#) illustrates the process.

Figure 19-5
Using a Lost-and-Found File



If the lost-and-found file appears on the computer that is running the primary database server after it restarts data replication, a transaction has been lost. The database server cannot reapply the transaction records in the lost-and-found file because conflicting updates might have occurred while the secondary database server was acting as a standard database server.

To reduce the risk of a lost transaction without running data replication in synchronous mode, use unbuffered logging for all the databases. This method reduces the amount of time between the writing and transfer of the transaction records from the primary database server to the secondary database server.

Threads That Handle HDR

The database server starts specialized threads to support data replication. As [Figure 19-6](#) shows, a thread called **drprsend** on the primary database server sends the contents of the HDR buffer across the network to a thread called **drsecrcv** on the secondary database server.

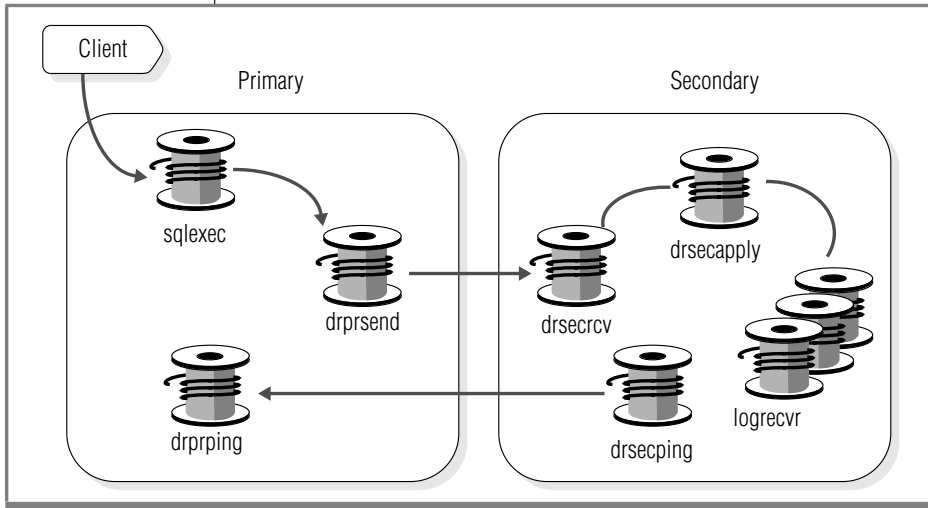


Figure 19-6
Threads That
Manage Data
Replication

A thread called **drsecapply** on the secondary database server copies the contents of the reception buffer to the recovery buffer. The **logrecvr** thread (or threads) performs logical recovery with the contents of the recovery buffer, applying the logical-log records to the dbspaces that the secondary database server manages. The ONCONFIG parameter `OFF_RECVRY_THREADS` specifies the number of **logrecvr** threads used.

The remaining threads that the database server starts for HDR are the **drprping** and **drsecping** threads, which are responsible for sending and receiving the messages that indicate if the two database servers are connected.

Checkpoints Between Database Servers

Checkpoints between database servers in a replication pair are synchronous, regardless of the value of DRINTERVAL. (See [“Checkpoints” on page 15-11.](#)) A checkpoint on the primary database server completes only after it completes on the secondary database server. If the checkpoint does not complete within the time that the ONCONFIG parameter DRTIMEOUT specifies, the primary database server assumes that a failure has occurred. See [“HDR Failures Defined” on page 19-17.](#)

How Data Synchronization Is Tracked

To keep track of synchronization, each database server in the pair keeps track of the following information in its archive reserved page:

- The ID of the logical-log file that contains the last completed checkpoint
- The position of the checkpoint record within the logical-log file
- The ID of the last logical-log file sent (or received)
- The page number of the last logical-log record sent (or received)

The database servers use this information internally to synchronize data replication.

HDR Failures

This section discusses the causes and consequences of an HDR failure, as well as the administrator's options for managing failure and restarting data replication.

HDR Failures Defined

An HDR failure is a loss of connection between the database servers in a replication pair. Any of the following situations might cause a data-replication failure:

- A catastrophic failure (such as a fire or large earthquake) at the site of one of the database servers
- A disruption of the networking cables that join the two database servers
- An excessive delay in processing on one of the database servers
- A disk failure on the secondary database server that is not resolved by a mirrored chunk

Tip: *An HDR failure does not necessarily mean that one of the database servers has failed, only that the HDR connection between the two database servers is lost.*



Detection of HDR Failures

The database server interprets either of the following conditions as an HDR failure:

- A specified time-out value was exceeded.
During normal HDR operation, a database server expects confirmation of communication from the other database server in the pair. Each database server in the pair has an ONCONFIG parameter, DRTIMEOUT, that specifies a number of seconds. If confirmation from the other database server in a pair does not return within the number of seconds that DRTIMEOUT specifies, the database server assumes that an HDR failure has occurred.
- The other database server does not respond to the periodic messaging (*pinging*) attempts over the network.
Both database servers send a ping to the other database server in the pair when the number of seconds specified by the DRTIMEOUT parameter on that database server has passed. The database servers ping each other regardless of whether the primary database server sends any records to the secondary database server. If a database server does not respond to four ping attempts in a row, the other database server assumes that an HDR failure has occurred.

Actions When an HDR Failure Is Detected

After a database server detects an HDR failure, it writes a message to its message log (for example, DR: receive error) and turns data replication off. If an HDR failure occurs, the HDR connection between the two database servers is dropped and the secondary database server remains in read-only mode.

Considerations After HDR Failure

Consider the following two issues when an HDR failure occurs:

- How the clients should react to the failure

If a real failure occurs (not just transitory network slowness or failure), you probably want clients that are using the failed database server to *redirect* to the other database server in the pair. For instruction on how to redirect clients, see [“Redirection and Connectivity for Data-Replication Clients” on page 19-21](#).

- How the database servers should react to the failure

Which administrative actions to take after an HDR failure depends on whether the primary database server or the secondary database server failed. For a discussion of this topic, see [“Actions to Take If the Secondary Database Server Fails” on page 19-19](#) and [“Actions to Take If the Primary Database Server Fails” on page 19-20](#).

If you redirect clients, consider what sort of load the additional clients place on the remaining database server. You might need to increase the space devoted to the logical log or back up the logical-log files more frequently.

Actions to Take If the Secondary Database Server Fails

If the secondary database server fails, the primary database server remains online.

To redirect clients that use the secondary database server to the primary database server, use any of the methods explained in [“Redirection and Connectivity for Data-Replication Clients” on page 19-21](#). If you redirect these clients, the primary database server might require an additional temporary dbspace for temporary tables and sorting.

You do not need to change the type of the primary database server to standard.

To restart data replication after a failure of the secondary database server, follow the steps in [“Restarting If the Secondary Database Server Fails” on page 20-31](#).

Actions to Take If the Primary Database Server Fails

If the primary database server fails, the secondary database server can behave in the following ways:

- The secondary database server can remain in logical-recovery mode. In other words, no action is taken. This would be the case if you expect the HDR connection to be restored very soon.
- Use *manual switchover* to change the database server mode to standard.

Manual Switchover

Manual switchover means that the administrator of the secondary database server changes the type of the secondary database server to standard. The secondary database server rolls back any open transactions and then comes into online mode as a standard database server, so that it can accept updates from client applications. For an explanation of how to perform the switchover, see [“Changing the Database Server Type” on page 20-21](#).

Restarting After a Manual Switchover

For a list of the steps involved in restarting data replication after a manual switchover, see [“The Secondary Database Server Is Changed to a Standard Database Server” on page 20-31](#).

Restarting If the Secondary Database Server Is Not Switched to Standard

If the secondary database server is not changed to type standard, follow the steps in [“The Secondary Database Server Was Not Changed to a Standard Database Server” on page 20-31](#).

Redirection and Connectivity for Data-Replication Clients

To connect to the database servers in a replication pair, clients use the same methods with which they connect to standard database servers. For an explanation of these methods, see the descriptions of the CONNECT and DATABASE statements in the *IBM Informix Guide to SQL: Syntax*.

After a failure of one of the database servers in a pair, you might want to *redirect* the clients that use the failed database server. (You might not want clients to be redirected. For example, if you anticipate that the database servers will be functioning again in a short time, redirecting clients might not be appropriate.)

The database server does not have a transparent mechanism for directing client requests to different database servers in a replication pair, but you can automate this action from within the application, as described in [“Handling Redirection Within an Application” on page 19-28](#). Some of the client connectivity drivers included in the IBM Informix Client Software Developer’s Kit have specific mechanisms for automating redirection. For details, see the IBM Informix Client Software Developer’s Kit documentation.

Designing Clients for Redirection

When you design client applications, you must make some decisions on redirection strategies. Specifically, you must decide whether to handle redirection within the application and which redirection mechanism to use. The three different redirection mechanisms are as follows:

- Automatic redirection with the **DBPATH** environment variable
- Administrator-controlled redirection with the connectivity information
- User-controlled redirection with the **INFORMIXSERVER** environment variable

The mechanism that you employ determines which CONNECT syntax you can use in your application. The following sections describe each of the three redirection mechanisms.

Directing Clients Automatically with DBPATH

This section explains the steps that you must follow to redirect clients with the **DBPATH** environment variable and the connectivity strategy that supports this method.

How the DBPATH Redirection Method Works

When an application does not explicitly specify a database server in the **CONNECT** statement, and the database server that the **INFORMIXSERVER** environment variable specifies is unavailable, the client uses the **DBPATH** environment variable to locate the database (and database server).

If one of the database servers in a replication pair is unusable, applications that use that database server need not reset their **INFORMIXSERVER** environment variable if their **DBPATH** environment variable is set to the other database server in the pair. Their **INFORMIXSERVER** environment variable should always contain the name of the database server that they use regularly, and their **DBPATH** environment variable should always contain the name of the alternative database server in the pair.

For example, if applications normally use a database server called **cliff_ol**, and the database server paired with **cliff_ol** in a replication pair is called **beach_ol**, the environment variables for those applications would be as follows:

```
INFORMIXSERVER cliff_ol
DBPATH         //beach_ol
```

Because the **DBPATH** environment variable is read only (if needed) when an application issues a **CONNECT** statement, applications must restart in order for redirection to occur.

An application can contain code that tests whether a connection has failed and, if so, attempts to reconnect. If an application has this code, you do not need to restart it.

You can use the `CONNECT TO database` statement with this method of redirection. For this method to work, you cannot use any of the following statements:

- `CONNECT TO DEFAULT`
- `CONNECT TO database@dbserver`
- `CONNECT TO @dbserver`

The reason for this restriction is that an application does not use `DBPATH` if a `CONNECT` statement specifies a database server. For more information about `DBPATH`, refer to the *IBM Informix Guide to SQL: Reference*.

What the Administrator Needs to Do

Administrators take no action to redirect clients, but they might need to attend to the type of the database server.

What the User Needs to Do

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities.

If your applications do not include such code, users who are running clients must quit and restart all applications.

Directing Clients with the Connectivity Information

This section explains the steps in redirecting clients with the connectivity information and the connectivity strategy that supports this method.

Operating System	Location of Connectivity Information
UNIX	The <code>INFORMIXSQLHOSTS</code> environment variable specifies the full pathname and filename of the connection information in <code>\$INFORMIXDIR/etc/sqlhosts</code> . For more information about <code>INFORMIXSQLHOSTS</code> , see the <i>IBM Informix Guide to SQL: Reference</i> .
Windows	The connectivity information is in a key in the Windows registry called <code>HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS</code> .

How Redirection with the Connectivity Information Works

The connectivity information-redirection method relies on the fact that when an application connects to a database server, it uses the connectivity information to find that database server.

If one of the database servers in a replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in [“Changing the Connectivity Information” on page 19-24](#), the fields of the unavailable database server (except for the **dbservername** field) are changed to point to the remaining database server in the replication pair.

Because the connectivity information is read when a CONNECT statement is issued, applications might need to restart for redirection to occur. Applications can contain code that tests whether a connection has failed and issues a reconnect statement, if necessary. If a connection has failed, redirection is automatic, and you do not need to restart applications for redirection to occur.

Applications can use the following connectivity statements to support this method of redirection:

- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

Applications can also use the following connectivity statements, provided that the **INFORMIXSERVER** environment variable always remains set to the same database server name and the **DBPATH** environment variable is not set:

- CONNECT TO DEFAULT
- CONNECT TO *database*

Changing the Connectivity Information

To use the connectivity information to redirect clients, you must change the connectivity information for the clients and change other connectivity files, if necessary.

For more information, refer to [“Configuring HDR Connectivity” on page 20-9](#) and [Chapter 3, “Client/Server Communications.”](#)

To change the connectivity information on the client computer

1. Comment out the entry for the failed database server.
2. Add an entry that specifies the `dbservername` of the failed database server in the **servername** field and information for the database server to which you are redirecting clients in the **nettype**, **hostname**, and **servicename** fields.
3. Use the following options in the **sqlhosts** file or registry to redirect applications to another database server if a failure should occur:
 1. [“Connection-Redirection Option” on page 3-42](#))
 2. [“End-of-Group Option” on page 3-43](#)
 3. [“Group Option” on page 3-44](#)

[Figure 19-7 on page 19-26](#) shows how connectivity values might be modified to redirect clients.

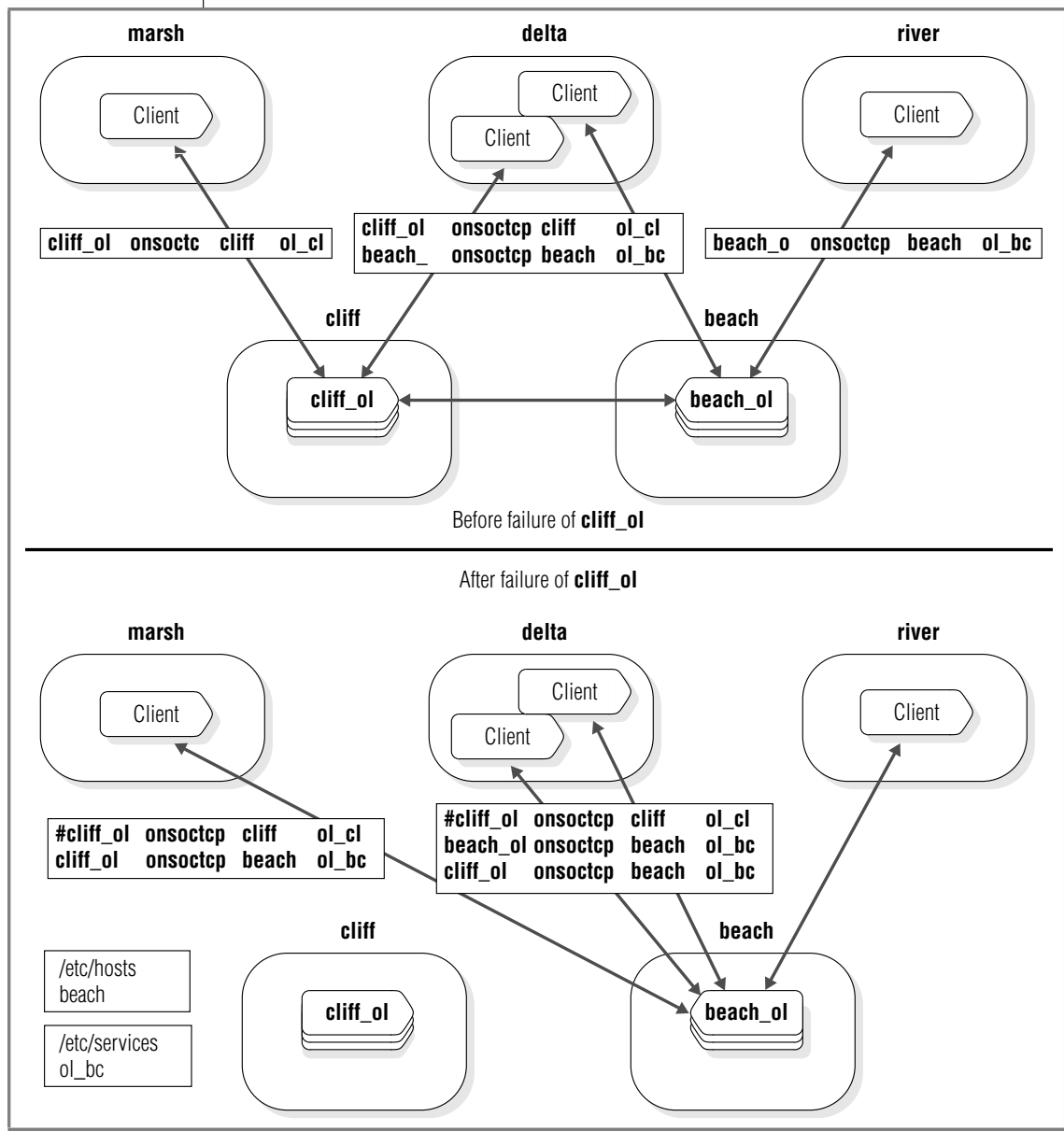
You do not need to change entries in the connectivity information on either of the computers that is running the database servers.

To change other connectivity files

You also must ensure that the following statements are true on the client computer before that client can reconnect to the other database server.

1. The **/etc/hosts** file on UNIX or **hosts** file on Windows has an entry for the **hostname** of the computer that is running the database server to which you are redirecting clients.
2. The **/etc/services** file on UNIX or **services** file on Windows has an entry for the **servicename** of the database server to which you are redirecting clients.

Figure 19-7
Connectivity Values Before and After a Failure of the cliff_ol Database Server



Connecting to the Database Server

After the administrator changes the connectivity information and other connectivity files (if needed), clients connect to the database server to which the administrator redirects them when they issue their next CONNECT statement.

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Directing Clients with INFORMIXSERVER

This section explains the steps in redirecting clients with the INFORMIXSERVER environment variable and the connectivity strategy that supports this method.

How Redirection Works with INFORMIXSERVER

The INFORMIXSERVER redirection method relies on the fact that when an application does not explicitly specify a database server in the CONNECT statement, the database server connects to the client that the INFORMIXSERVER environment variable specifies.

If one of the database servers in a replication pair is unusable, applications that use that database server can reset their INFORMIXSERVER environment variable to the other database server in the pair to access the same data.

Applications read the value of the INFORMIXSERVER environment variable only when they start. Therefore, applications must be restarted to recognize a change in the environment variable.

To support this method of redirection, you can use the following connectivity statements:

- CONNECT TO DEFAULT
- CONNECT TO *database*

You cannot use the `CONNECT TO database@dbserver` or `CONNECT TO @dbserver` statements for this method. When a database server is explicitly named, the `CONNECT` statement does not use the `INFORMIXSERVER` environment variable to find a database server.

What the Administrator Needs to Do

Administrators take no action to redirect the clients, but they might need to change the type of the database server.

What the User Needs to Do

Users who are running client applications must perform the following three steps when they decide to redirect clients with the `INFORMIXSERVER` environment variable.

To redirect clients with the `INFORMIXSERVER` environment variable

1. Quit their applications.
2. Change their `INFORMIXSERVER` environment variable to hold the name of the other database server in the replication pair.
3. Restart their applications.

Handling Redirection Within an Application

If you use `DBPATH` or connectivity information to redirect, you can include in your clients a routine that handles errors when clients encounter an HDR failure. The routine can call another function that contains a loop that tries repeatedly to connect to the other database server in the pair. This routine redirects clients without requiring the user to exit the application and restart it.

Figure 19-8 shows an example of a function in a client application using the **DBPATH** redirection mechanism that loops as it attempts to reconnect. Once it establishes a connection, it also tests the type of the database server to make sure it is not a secondary database server. If the database server is still a secondary type, it calls another function to alert the user (or database server administrator) that the database server cannot accept updates.

```
/* The routine assumes that the INFORMIXSERVER environment
 * variable is set to the database server that the client
 * normally uses and that the DBPATH environment variable
 * is set to the other database server in the pair.
 */

#define SLEEPTIME 15
#define MAXTRIES 10

main()
{
    int connected = 0;
    int tries;
    for (tries = 0; tries < MAXTRIES && connected == 0; tries++)
    {
        EXEC SQL CONNECT TO "superstores";
        if (strcmp(SQLSTATE, "00000"))
        {
            if (sqlca.sqlwarn.sqlwarn6 != 'W')
            {
                notify_admin();
                if (tries < MAXTRIES - 1)
                    sleep(SLEEPTIME);
            }
            else connected = 1;
        }
    }
    return ((tries == MAXTRIES)? -1:0);
}
```

Figure 19-8
*Example of a
CONNECT Loop for
DBPATH Redirection
Mechanism*

This example assumes the **DBPATH** redirection mechanism and uses a form of the **CONNECT** statement that supports the **DBPATH** redirection method. If you used the connectivity information to redirect, you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "superstores@cliff_ol";
```

In this example, **superstores@cliff_ol** refers to a database on a database server that the client computer recognizes. For redirection to occur, the administrator must change the connectivity information to make that name refer to a different database server. You might need to adjust the amount of time that the client waits before it tries to connect or the number of tries that the function makes. Provide enough time for an administrative action on the database server (to change the connectivity information or change the type of the secondary database server to standard).

Comparing Different Redirection Mechanisms

Figure 19-9 summarizes the differences among the three redirection mechanisms.

Figure 19-9
Comparison of Redirection Methods for Different Connectivity Strategies

	DBPATH		Connectivity Information		INFORMIXSERVER
	Automatic Redirection	User Redirection	Automatic Redirection	User Redirection	User Redirection
When is a client redirected?	When the client next tries to connect with a specified database		After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server		When the client restarts and reads a new value for the INFORMIXSERVER environment variable
Do clients need to be restarted to be redirected?	No	Yes	No	Yes	Yes
What is the scope of the redirection?	Individual clients redirected	Individual clients redirected	All clients that use a given database server redirected	Individual clients redirected	Individual clients redirected
Are changes to environment variables required?	No		No		Yes

Designing HDR Clients

This section discusses various design considerations (in addition to the redirection considerations discussed earlier) for clients that connect to database servers that are running data replication.

Setting Lock Mode to Wait for Access to Primary Database Server

When the database server performs a logical recovery, it normally defers index builds until the end of the recovery. However, if the database server is acting as a secondary database server, it is in logical recovery mode for as long as data replication is running. Thus, secondary database servers must use a different mechanism to perform index builds.

The mechanism used is as follows. When the secondary database server receives a logical-log record that necessitates a corresponding index build, it sends a message back to the primary database server to request a physical copy of the index. The primary database server has a lock on the table that is being updated. The owner of the lock is a **dr_btse** thread. The application thread that is executing is free to continue processing. The **dr_btse** thread cannot release the lock, however, until the secondary database server acknowledges receipt of the index. If the application tries to access the table while it is locked, this attempt fails, unless the application has set the lock mode to wait.

If applications do not have lock mode set to WAIT, some unexpected errors might occur. For example, many SQL statements cause updates to the catalog table indexes. The following sequence of SQL statements fails if the lock mode of the application is not set to WAIT:

```
CREATE DATABASE db_name;  
DATABASE db_name;  
CREATE TABLE tab_name;
```

These SQL statements would fail because the CREATE DATABASE statement creates indexes on the **systables** catalog table and, therefore, places a lock on the table until the indexes are copied over to the secondary database server. Meanwhile, the CREATE TABLE statement tries to insert a row into the **systables** catalog table. The insert fails, however, because the table is locked.

This application would fail because both the CREATE DATABASE and CREATE TABLE statements cause updates to the **systables** catalog table index.

Designing Clients to Use the Secondary Database Server

To achieve a degree of load balancing when you use data replication, have some client applications use the secondary database server in a data-replication pair. Design all client applications that use the secondary database server with the following points in mind:

- Any SQL statements that attempt to modify data fail.
- Locking and isolation levels are not the same as on the standard database server.
- Temporary dbspaces must be used for sorting and temporary tables.

The following sections discuss these considerations in more detail.

No Data Modification Statements

SQL statements that update dbspaces in logical-recovery mode on the secondary database server are not allowed. For example, the ALTER FRAGMENT and CREATE DATABASE statements would produce errors. For a complete list of data modification statements, see the *IBM Informix Guide to SQL: Syntax*.

To prevent clients that are using the secondary database server from issuing updating statements, you can take either of the following actions:

- Write client applications that do not issue updating statements.
- Make all updating statements conditional.

To make statements that perform an update conditional, make sure that client applications test **sqlwarn6** of the **sqlwarn** field in the ESQL/C **sqlca** structure (and equivalent values for other SQL APIs). The database server sets **sqlwarn6** to **w** when it runs as a secondary database server.

Locking and Isolation Level

Because all clients that use the secondary database server only read data, locking to ensure isolation between those clients is not required. However, a client that uses the secondary database server is not protected from the activity of users on the primary database server because the **logrecvr** threads that perform logical recovery do not use locking.

For example, if a client connected to the secondary database server reads a row, nothing prevents a user on the primary database server from updating that row, even if the client connected to the secondary database server has issued a SET ISOLATION TO REPEATABLE READ statement. The update is reflected on the secondary database server as the logical-log records for the committed transaction are processed. Thus, all queries on the secondary database server are essentially dirty with respect to changes that occur on the primary database server, even though a client that uses the secondary database server might explicitly set the isolation level to something other than Dirty Read.



Important: All queries on the secondary database server are dirty reads. Do not run queries on the secondary database server while log records that contain data definition language (DDL) statements are being replicated and applied on the secondary database server.

To perform DDL operations on the primary database server

1. Stop all activity (such as transactions) on the secondary database server.
2. Run the DDL statements on the primary database server.
For a complete list of DDL statements, see the *IBM Informix Guide to SQL: Syntax*.
3. To issue a checkpoint on the primary database server, use the **onmode -c** command.
4. Wait for this checkpoint to get replicated on the secondary database server.
5. Resume activity on the secondary database server.

Use of Temporary Dbspaces for Sorting and Temporary Tables

Even though the secondary database server is in read-only mode, it does write when it needs to perform a sort or create a temporary table. “[Temporary Dbspaces](#)” on [page 9-19](#) explains where the database server finds temporary space to use during a sort or for a temporary table. To prevent the secondary database server from writing to a dbspace that is in logical-recovery mode, you must take the following actions:

- Ensure that one or more temporary dbspaces exist. For instructions on creating a temporary dbspace, see “[Creating a Dbpace](#)” on [page 10-14](#).
- Then, choose one of the following actions:
 - Set the DBSPACETEMP parameter in the ONCONFIG file of the secondary database server to the temporary dbspace or dbspaces.
 - Set the DBSPACETEMP environment variable of the client applications to the temporary dbspace or dbspaces.

Using High-Availability Data Replication

In This Chapter	20-3
Planning for HDR	20-3
Configuring a System for HDR.	20-4
Meeting Hardware and Operating-System Requirements	20-4
Meeting Database and Data Requirements	20-5
Meeting Database Server Configuration Requirements	20-5
Database Server Version	20-6
Storage Space and Chunk Configuration.	20-6
Mirroring	20-7
Physical-Log Configuration	20-7
Dbospace and Logical-Log Tape Backup Devices	20-7
Logical-Log Configuration	20-8
Shared-Memory Configuration	20-8
HDR Parameters	20-8
Configuring HDR Connectivity	20-9
Starting HDR for the First Time	20-9
Performing Basic Administration Tasks.	20-14
Changing Database Server Configuration Parameters	20-14
Backing Up Storage Spaces and Logical-Log Files	20-15
Changing the Logging Mode of Databases	20-15
Adding and Dropping Chunks and Storage Spaces	20-16
Renaming Chunks.	20-16
Saving Chunk Status on the Secondary Database Server.	20-17
Using and Changing Mirroring of Chunks	20-18
Managing the Physical Log.	20-19
Managing the Logical Log	20-19

Managing Virtual Processors	20-19
Managing Shared Memory	20-20
Changing the Database Server Mode	20-20
Changing the Database Server Type	20-21
Monitoring HDR Status	20-22
Using Command-Line Utilities	20-22
Using SMI Tables	20-24
Using ON-Monitor	20-24
Restoring Data After Media Failure Occurs	20-24
Restoring After a Media Failure on the Primary Database Server	20-25
Restoring After a Media Failure on the Secondary Database Server	20-26
Restarting HDR After a Failure	20-27
Restarting After Critical Data Is Damaged	20-27
Critical Media Failure on the Primary Database Server	20-28
Critical Media Failure on the Secondary Database Server	20-29
Critical Media Failure on Both Database Servers	20-29
Restarting If Critical Data Is Not Damaged	20-30
Restarting After a Network Failure	20-30
Restarting If the Secondary Database Server Fails	20-31
Restarting If the Primary Database Server Fails	20-31

In This Chapter

This chapter describes how to plan, configure, start, and monitor High-Availability Data Replication (HDR) for Dynamic Server, and how to restore data after a media failure. If you plan to use HDR, read this entire chapter first. If you plan to use IBM Informix Enterprise Replication, see the *IBM Informix Dynamic Server Enterprise Replication Guide*.

[Chapter 19, “High-Availability Data Replication,”](#) explains what HDR is, how it works, and how to design client applications for an HDR environment.

Planning for HDR

Before you start setting up computers and database servers to use HDR, you might want to do some initial planning. The following list contains planning tasks to perform:

- Choose and acquire appropriate hardware.
- If you are using more than one database server to store data that you want to replicate, migrate and redistribute this data so that it can be managed by a single database server.
- Ensure that all databases you want to replicate use transaction logging. To turn on transaction logging, see [Chapter 12, “Managing Database-Logging Mode.”](#)
- Develop client applications to make use of both database servers in the replication pair. For a discussion of design considerations, refer to [“Redirection and Connectivity for Data-Replication Clients” on page 19-21](#) and [“Designing Clients to Use the Secondary Database Server” on page 19-33](#).
- Create a schedule for starting HDR for the first time.

- Design a storage-space and logical-log backup schedule for the primary database server.
- Produce a plan for how to handle failures of either database server and how to restart HDR after a failure. Read [“Redirection and Connectivity for Data-Replication Clients”](#) on page 19-21.

Configuring a System for HDR

To configure your system for HDR, you must take the following actions:

- Meet hardware and operating-system requirements.
- Meet database and data requirements.
- Meet database server configuration requirements.
- Configure HDR connectivity.

Each of these topics is explained in this section.

Meeting Hardware and Operating-System Requirements

For an HDR database server pair to function, it must meet the following hardware requirements:

- The computers that run the primary and secondary database servers must be identical (same vendor and architecture).
- The operating systems on the computers that run the primary and secondary database servers must be identical.
- The hardware that runs the primary and secondary database servers must support network capabilities.
- The amount of disk space allocated to dbspaces for the primary and secondary database servers must be equal. The type of disk space is irrelevant; you can use any mixture of raw or cooked spaces on the two database servers.

Meeting Database and Data Requirements

For an HDR database server pair to function, you must meet the following database and data requirements:

- **All data must be logged**

All databases that you want to replicate must have transaction logging turned on.

This requirement is important because the secondary database server uses logical-log records from the primary database server to update the data that it manages. If databases managed by the primary database server do not use logging, updates to those databases do not generate log records, so the secondary database server has no means of updating the replicated data. Logging can be buffered or unbuffered.

If you need to turn on transaction logging before you start HDR, see [“Turning On Transaction Logging with ontape” on page 12-7](#).

- **The data must reside in dbspaces or sbspaces**

If your primary database server has simple large objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal HDR processing. However, simple-large-object data within dbspaces is replicated.

Smart large objects, which are stored in sbspaces, are replicated. The sbspaces must be logged. User-defined types (UDTs) are replicated, unless they have out-of-row data stored in operating system files.

Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server.

Meeting Database Server Configuration Requirements

For an HDR database server pair to function, you must fully configure each of the database servers. For information on configuring a database server, refer to [“Installing and Configuring the Database Server” on page 1-1](#). You can then use the relevant aspects of that configuration to configure the other database server in the pair. For more information on the configuration parameters, see the *Administrator’s Reference*.

This section describes the following configuration considerations for HDR database server pairs:

- [Database Server Version](#)
- [Storage Space and Chunk Configuration](#)
- [Mirroring](#)
- [Physical-Log Configuration](#)
- [Dbpace and Logical-Log Tape Backup Devices](#)
- [Logical-Log Configuration](#)
- [Shared-Memory Configuration](#)
- [HDR Parameters](#)

Database Server Version

The versions of the database server on the primary and secondary database servers must be identical.

Storage Space and Chunk Configuration

The number of dbspaces, the number of chunks, their sizes, their pathnames, and their offsets must be identical on the primary and secondary database servers.

It is recommended that the configuration contain at least one temporary dbpace. See [“Use of Temporary Dbspaces for Sorting and Temporary Tables”](#) on page 19-35.

You should use symbolic links for the chunk pathnames, as explained in [“Allocating Raw Disk Space on UNIX”](#) on page 10-9.

Important: *If you do not use symbolic links for chunk pathnames, you cannot easily change the pathname of a chunk. For more information, see [“Renaming Chunks”](#) on page 20-16.* ♦

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME
- ROOTOFFSET

UNIX



- ROOTPATH
- ROOTSIZE

Mirroring

You do not have to set the MIRROR parameter to the same value on the two database servers; you can enable mirroring on one database server and disable mirroring on the other. However, if you specify a mirrored chunk for the root chunk of the primary database server, you must also specify a mirrored chunk for the root chunk on the secondary database server. Therefore, the following ONCONFIG parameters must be set to the same value on both database servers:

- MIRROROFFSET
- MIRRORPATH

Physical-Log Configuration

The physical log should be identical on both database servers. The following ONCONFIG parameters must have the same value on each database server:

- PHYSDBS
- PHYSFILE

Dbospace and Logical-Log Tape Backup Devices

You can specify different tape devices for the primary and secondary database servers.

If you use ON-Bar, set the ON-Bar configuration parameters to the same value on both database servers. For information on the ON-Bar parameters, see the *IBM Informix Backup and Restore Guide*.

If you use **ontape**, the tape size and tape block size for the storage-space and logical-log backup devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK
- LTAPESIZE

- TAPEBLK
- TAPESIZE

To use a tape to its full physical capacity, set LTAPESIZE and TAPESIZE to 0.

Logical-Log Configuration

All log records are replicated to the secondary server. You must configure the same number of logical-log files and the same logical-log size for both database servers. The following ONCONFIG parameters must have the same value on each database server:

- LOGFILES
- LOGSIZE
- DYNAMIC_LOGS

The database server logs the addition of logical-log files. Logical-log files added dynamically on the primary server are automatically replicated on the secondary server. Although the DYNAMIC_LOGS value on the secondary server has no effect, keep DYNAMIC_LOGS in sync with the value on the primary server, in case their roles switch.

Shared-Memory Configuration

Set all the shared-memory configuration parameters (SHMADD, SHMTOTAL, SHMVIRTSIZE) to the same values on the two database servers.

HDR Parameters

The following HDR parameters must be set to the same value on both database servers in the replication pair:

- DRINTERVAL
- DRLOSTFOUND
- DRTIMEOUT

Configuring HDR Connectivity

For an HDR database server pair to function, the two database servers must be able to establish a connection with one another. To satisfy this requirement, the connectivity information on each of the computers that is running the database server in a replication pair must have at least the following entries:

- An entry that identifies the database server that is running on that computer
- An entry that identifies the other database server in the data-replication pair



Important: Set the *nettype* field of the *sqlhosts* file or registry and the *NETTYPE* configuration parameter to a network protocol such as *ontlittcp*, *onsoctcp*, or *ontlispix* so that the database servers on two different computers can communicate with each other. HDR does not work if the *nettype* field specifies a non-network protocol such as *onipcshm*, *onipcstr*, or *onipcnmp*.

For information on how to redirect clients and change connectivity information, see [“Redirection and Connectivity for Data-Replication Clients” on page 19-21](#).

Starting HDR for the First Time

After you complete the HDR configuration, you are ready to start HDR. This section describes the necessary steps for starting HDR.

Suppose you want to start HDR on two database servers, **ServerA** and **ServerB**. The procedure for starting HDR, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. [Figure 20-1 on page 20-12](#) lists the commands required to perform each step and the messages sent to the message log. You can use **ontape** or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure, however.



Important: If you use ON-Bar to perform the backup and restore, **ontape** is required on both database servers. You cannot remove **ontape** from database servers participating in HDR.



Tip: For the procedure on how to start HDR using external backup and restore, see the “IBM Informix Backup and Restore Guide.”

To start HDR

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.
2. Create a level-0 backup of **ServerA**.
3. Use the **onmode -d** command to set the type of **ServerA** to primary and to indicate the name of the associated secondary database server (in this case **ServerB**).

When you issue an **onmode -d** command, the database server attempts to establish a connection with the other database server in the HDR pair and to start HDR operation. The attempt to establish a connection succeeds only if the other database server in the pair is already set to the correct type.

At this point, **ServerB** is not online and is not set to type secondary, so the HDR connection is not established.

4. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 1. Do not perform a logical restore.

If you are using **ontape**, use the **ontape -p** option. You cannot use the **ontape -r** option because it performs both a physical and a logical restore.

If you are using ON-Bar, use the **onbar -r -p** command to perform a physical restore.

5. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server.

ServerB tries to establish an HDR connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

Before HDR begins, the secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 1. If all these logical-log records still reside on the primary database server disk, the primary database server sends these records directly to the secondary database server over the network, and logical recovery occurs automatically.

If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 6.



Important: You must complete steps 4 and 5 together. If you need to shut down and restart the secondary database server after step 4, you must redo step 4.

6. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.

If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

Figure 20-1
Steps to Start HDR for the First Time

Step	On the Primary	On the Secondary
1	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2	ontape command ontape -s -L 0 ON-Bar command onbar -b -L 0 Messages to message log Level 0 archive started on rootdbs. Archive on rootdbs completed.	ontape command ontape -p Answer no when you are prompted to back up the logs. ON-Bar command onbar -r -p Messages to message log IBM Informix Database Server Initialized -- Shared Memory Initialized Recovery Mode Physical restore of rootdbs started. Physical restore of rootdbs completed.
3		

Step	On the Primary	On the Secondary
4		<p>onmode command</p> <pre>onmode -d secondary prim_name</pre> <p>Messages to message log</p> <pre>DR: new type = secondary, primary server name = prim_name</pre> <p>If all the logical-log records written to the primary database server since step 1 still reside on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 5 must be performed).</p>
	<p>Messages to message log</p> <pre>DR: Primary server connected DR: Primary server operational</pre>	<p>Messages to message log</p> <pre>DR: Trying to connect to primary server DR: Secondary server connected DR: Failure recovery from disk in process. n recovery worker threads will be started. Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xn timer DR: Secondary server operational</pre>

(2 of 3)

Step	On the Primary	On the Secondary
5		ontape command ontape -l ON-Bar command onbar -l
	Messages to message log DR: Primary server connected DR: Primary server operational	Messages to message log DR: Secondary server connected DR: Failure recovery from disk in process. n recovery worker threads will be started. Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xxxxxxx DR: Secondary server operational

(3 of 3)

Performing Basic Administration Tasks

This section contains instructions on how to perform database server administration tasks once your system is running HDR.

Changing Database Server Configuration Parameters

Some of the configuration parameters must be set to the same value on both database servers in the replication pair (as listed under [“Meeting Database Server Configuration Requirements” on page 20-5](#)). Other Dynamic Server configuration parameters can be set to different values.

If you need to change a configuration parameter that must have the same value on both database servers, you must change the value of that parameter in the ONCONFIG file of both database servers.

To make changes to ONCONFIG files

1. Bring each database server offline with the **onmode -k** option.

2. Change the parameters on each database server.
3. Starting with the last database server that you brought offline, bring each database server back online.

For example, if you brought the secondary database server offline last, bring the secondary database server online first. [Figure 20-1 on page 20-12](#) and [Figure 20-1 on page 20-12](#) list the procedures for bringing the primary and secondary database servers back online.

If the configuration parameter does not need to have the same value on each database server in the replication pair, you can change the value on the primary or secondary database server individually.

Backing Up Storage Spaces and Logical-Log Files

When you use HDR, you must back up logical-log files and storage spaces only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.

You must use the same backup and restore tool on both database servers.

The block size and tape size used (for both storage-space backups and logical-log backups) must be identical on the primary and secondary database servers.

You can use **ontape** to set the tape size to 0 to automatically use the full physical capacity of a tape.

Changing the Logging Mode of Databases

You cannot turn on transaction logging for databases on the primary database server while you are using HDR. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

To turn on database logging

1. To turn HDR off, shut down the secondary database server.

2. Turn on database logging.

After you turn on logging for a database, if you start data replication without performing the level-0 backup on the primary database server and restore on the secondary database server, the database on the primary and secondary database servers might have different data. This situation could cause data-replication problems.

3. Perform a level-0 backup on the primary database server and restore on the secondary database server. This procedure is described in [“Starting HDR for the First Time” on page 20-9](#).

Adding and Dropping Chunks and Storage Spaces

You can perform disk-layout operations, such as adding or dropping chunks and dbspaces, only from the primary database server. The operation is replicated on the secondary database server. This arrangement ensures that the disk layout on both database servers in the replication pair remains consistent.

The directory pathname or the actual file for chunks must exist before you create them. Make sure the pathnames (and offsets, if applicable) exist on the secondary database server before you create a chunk on the primary database server, or else the database server turns off data replication.

Renaming Chunks

If you use symbolic links for chunk pathnames, you can rename chunks while HDR is operating. For instructions on renaming chunks, see the *IBM Informix Backup and Restore Guide*.

If you do not use symbolic links for chunk pathnames, you must take both database servers offline while renaming the chunks, for the time that it takes to complete a cold restore of the database server.

To rename chunks on a failed HDR server

1. Change the mode of the undamaged server to standard.
2. Take a level-0 backup of the standard server.
3. Shut down the standard server.
4. Rename the chunks on the standard server during a cold restore from the new level-0 archive (for instructions, see the *IBM Informix Backup and Restore Guide*).
5. Start the standard server.
6. Take another level-0 archive of the standard server.
7. Restore the failed server with the new level-0 backup and reestablish the HDR pair.

Saving Chunk Status on the Secondary Database Server

For a data-replication pair, if the status of a chunk (*down*, *online*) is changed on the secondary database server, and that secondary server is restarted before a checkpoint is completed, the updated chunk status is not saved.

To ensure that the new chunk status is flushed to the reserved pages on the secondary database server, force a checkpoint on the primary database server and verify that a checkpoint also completes on the secondary database server. The new chunk status is now retained even if the secondary database server is restarted.

If the primary chunk on the secondary database server is down, you can recover the primary chunk from the mirror chunk.

To recover the primary chunk from the mirror chunk

1. Run **onspaces -s** on the secondary database server to bring the primary chunk online.
You also can use ISA to bring the primary chunk online.
2. Run **onmode -c** on the primary database server to force a checkpoint.
3. Run **onmode -m** on the primary database server to verify that a checkpoint was actually performed.
4. Run **onmode -m** on the secondary database server to verify that a checkpoint was also completed on the secondary database server.

Once you complete these steps, the primary chunk will be online when you restart the secondary database server.

Using and Changing Mirroring of Chunks

You do not have to set the MIRROR configuration parameter to the same value on both database servers in the replication pair. In other words, you can enable or disable mirroring on either the primary or the secondary database server independently.

Before you can add a mirrored chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. If you want to mirror a dbspace on one of the database servers in the replication pair, you must create mirrored chunks for that dbspace on *both* database servers. For general information on allocating disk space, see [“Allocating Disk Space” on page 10-6](#).

You can perform disk-layout operations from the primary database server only. Thus, you can add or drop a mirrored chunk only from the primary database server. A mirrored chunk that you add to or drop from the primary database server is added to or dropped from the secondary database server as well. You must perform mirror recovery for the newly added mirror chunk on the secondary database server. For more information, see [“Recovering a Mirrored Chunk” on page 18-11](#).

When you drop a chunk from the primary database server, Dynamic Server automatically drops the corresponding chunk on the secondary database server. This applies to both primary and mirror chunks.

When you turn mirroring off for a dbspace on the primary database server, Dynamic Server does not turn mirroring off for the corresponding dbspace on the secondary database server. To turn off mirroring for a dbspace on the secondary database server independent of the primary server, use **onspaces -r**. For more information about turning off mirroring, see [“Ending Mirroring” on page 18-11](#).

You can take down a mirrored chunk or recover a mirrored chunk on either the primary or secondary database server. These processes are transparent to HDR.

Managing the Physical Log

The size of the physical log must be the same on both database servers. If you change the size and location of the physical log on the primary database server, this change is replicated to the secondary database server; however, the `PHYSDBS` and `PHYSFILE` parameters in the secondary `ONCONFIG` file are not updated. You must change these parameters manually by editing the `ONCONFIG` file. For the procedure for making this change, see [“Changing Database Server Configuration Parameters” on page 20-14](#).

For information on changing the size and location of the physical log, refer to [Chapter 16, “Managing the Physical Log.”](#)

Managing the Logical Log

The size of the logical log must be the same on both database servers. You can add or drop a logical-log file with the `onparams` utility, as described in [Chapter 14, “Managing Logical-Log Files.”](#) Dynamic Server replicates this change on the secondary database server; however, the `LOGFILES` parameter on the secondary database server is not updated. After you issue the `onparams` command from the primary database server, therefore, you must manually change the `LOGFILES` parameter to the desired value on the secondary database server. Finally, for the change to take effect, you must perform a level-0 backup of the root dbspace on the primary database server.

If you add a logical-log file to the primary database server, this file is available for use and flagged `F` as soon as you perform the level-0 backup. The new logical-log file on the secondary database server is still flagged `A`. However, this condition does not prevent the secondary database server from writing to the file.

Managing Virtual Processors

The number of virtual processors has no effect on data replication. You can configure and tune each database server in the pair individually.

Managing Shared Memory

If you make changes to the shared-memory ONCONFIG parameters on one database server, you must make the same changes at the same time to the shared-memory ONCONFIG parameters on the other database server. For the procedure for making this change, see [“Changing Database Server Configuration Parameters” on page 20-14](#).

Changing the Database Server Mode

To change the database server mode, use the **onmode** utility from the command line or ISA. For information about **onmode**, see the utilities chapter of the *IBM Informix Dynamic Server Administrator's Reference*.

[Figure 20-2](#) summarizes the effects of changing the mode of the primary database server.

Figure 20-2
Mode Changes on the Primary Database Server

On the Primary	On the Secondary	To Restart HDR
Any mode offline (onmode -k)	Secondary displays: DR: Receive error. HDR is turned off. The mode remains read-only.	Treat it like a failure of the primary. Two different scenarios are possible, depending on what you do with the secondary while the primary is offline: “The Secondary Database Server Was Not Changed to a Standard Database Server” on page 20-31 “The Secondary Database Server Is Changed to a Standard Database Server” on page 20-31
online or quiescent (onmode-s / onmode-u)	Secondary does not receive errors. HDR remains on. Mode remains read-only.	Use onmode -m on the primary.

Figure 20-3 summarizes the effects of changing the mode of the secondary database server.

Figure 20-3
Mode Changes on the Secondary Database Server

On the Secondary	On the Primary	To Restart HDR
Read-only offline (onmode -k)	Primary displays: DR: Receive error. HDR is turned off.	Treat it as you would a failure of the secondary. Follow the procedures in “Restarting If the Secondary Database Server Fails” on page 20-31.

Changing the Database Server Type

You can change the type of either the primary or the secondary database server.

You can change the database server type from secondary to standard only if HDR is turned off on the secondary database server. HDR is turned off when the data replication connection to the primary database server breaks or data replication fails on the secondary database server. When you take the standard database server offline and bring it back online, it does not attempt to connect to the other database server in the replication pair.

Use the following commands to switch the type:

- **hdrmksec.[sh | bat]** and **hdrmkpri.[sh | bat]** scripts

To switch the database server type using hdrmkpri and hdrmksec scripts

1. Shut down the primary database server (**ServerA**):
`onmode -ky`
2. With the secondary database server (**ServerB**) online, run the **hdrmkpri.sh** script on UNIX or **hdrmkpri.bat** script on Windows. Now **ServerB** is a primary database server.
3. For **ServerA**, run the **hdrmksec.sh** script on UNIX or **hdrmksec.bat** script on Windows. Now **ServerA** is a secondary database server.
4. Bring **ServerB** (primary database server) online.

Monitoring HDR Status

Monitor the HDR status of a database server to determine the following information:

- The database server type (primary, secondary, or standard)
- The name of the other database server in the pair
- Whether HDR is on
- The values of the HDR parameters

Using Command-Line Utilities

The header information displayed every time you execute **onstat** has a field to indicate if a database server is operating as a primary or secondary database server.

The following example shows a header for a database server that is the primary database server in a replication pair, and in online mode:

```
IBM Informix Dynamic Server Version 9.30.UC1      -- online(Prim) -- Up
45:08:57
```

This example shows a database server that is the secondary database server in a replication pair, and in read-only mode.

```
IBM Informix Dynamic Server Version 9.30.UC1      -- Read-Only (Sec) -- Up
45:08:57
```

The following example shows a header for a database server that is not involved in HDR. The type for this database server is standard.

```
IBM Informix Dynamic Server Version 9.30.UC1      -- online -- Up 20:10:57
```

onstat -g dri

To obtain full HDR monitoring information, execute the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The HDR state (on or off)
- The paired database server

- The last HDR checkpoint
- The values of the HDR configuration parameters

Example output is shown in [Figure 20-4](#). This example shows a primary database server, paired with a secondary database server that has the DBSERVERNAME of **beach_ol**. HDR has been started.

Data Replication:			
Type	State	Paired server	Last DR CKPT (id/pg)
primary	off	beach_ol	4/741
DRINTERVAL	30		
DRTIMEOUT	300		
DRLOSTFOUND	/usr/informix/etc/dr.lostfound		

Figure 20-4
onstat -g dri Output

oncheck -pr

If your database server is running HDR, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that HDR uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in [Figure 20-5](#).

Validating Informix Database Server reserved pages - PAGE_1ARCH & PAGE_2ARCH Using archive page PAGE_1ARCH.	
Archive Level	0
Real Time Archive Began	01/11/95 16:54:07
Time Stamp Archive Began	11913
Logical Log Unique Id	3
Logical Log Position	b018
DR Ckpt Logical Log Id	3
DR Ckpt Logical Log Pos	80018
DR Last Logical Log Id	3
DR Last Logical Log Page	128

Figure 20-5
*oncheck -pr
PAGE_1ARCH
Output for Database
Server Running
HDR*

Using SMI Tables

The **sysdri** table, described in the chapter on the **sysmaster** database in the *Administrator's Reference*, contains the following columns.

Column	Description
type	HDR server type
state	HDR server state
name	Database server name
intvl	HDR buffer flush interval
timeout	Network timeout
lostfound	HDR lost+found pathname

Using ON-Monitor

Choose Status→Replication to see information about HDR. This option displays the same information as the **onstat -g dri** option.

Restoring Data After Media Failure Occurs

The result of a disk failure depends on whether the disk failure occurs on the primary or the secondary database server, whether the chunks on the disk contain critical media (the root dbspace, a logical-log file, or the physical log), and whether the chunks are mirrored.

Restoring After a Media Failure on the Primary Database Server

[Figure 20-6 on page 20-26](#) summarizes the various scenarios for restoring data if the primary database server suffers media failure. The following issues are relevant:

- If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.
- In cases where the chunks are not mirrored, the procedure for restoring the primary database server depends on whether the disk that failed contains critical media.

If the disk contains critical media, the primary database server fails. You have to perform a full restore using the primary dbspace backups (or the secondary dbspace backups if the secondary database server was switched to standard mode and activity redirected). See [“Restarting After Critical Data Is Damaged” on page 20-27](#).

If the disk does not contain critical media, you can restore the affected dbspaces individually with a warm restore. A warm restore consists of two parts: first a restore of the failed dbspace from a backup and next a logical restore of all logical-log records written since that dbspace backup. For more information on performing a warm restore, see the *IBM Informix Backup and Restore Guide*. You must back up all logical-log files before you perform the warm restore.

Figure 20-6
Scenarios for Media Failure on the Primary Database Server

HDR Server	Critical Media	Chunks Mirrored	Effect of Failure and Procedure for Restoring Media
Primary	Yes	No	Primary database server fails. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 20-27.
Primary	Yes	Yes	Primary database server remains online. Follow the procedures in “Recovering a Mirrored Chunk” on page 18-11.
Primary	No	No	Primary database server remains online. Follow the procedure in your Dynamic Server backup and restore manual for performing a warm restore of a dbspace from a dbspace backup. Back up all logical-log files before you perform the warm restore.
Primary	No	Yes	Primary database server remains online. Follow the procedures in “Recovering a Mirrored Chunk” on page 18-11.

Restoring After a Media Failure on the Secondary Database Server

[Figure 20-7](#) summarizes the various scenarios for restoring data if the secondary database server suffers media failure. The following issues are relevant:

- If chunks are mirrored, you can perform recovery just as you would for a standard database server that uses mirroring.
- In cases where the chunks are not mirrored, the secondary database server fails if the disk contains critical media but remains online if the disk does not contain critical media. In both cases, you have to perform a full restore using the dbspace backups on the primary database server. (See [“Restarting After Critical Data Is Damaged”](#) on [page 20-27](#).) In the second case, you cannot restore selected dbspaces from the secondary dbspace backup because they might now deviate from the corresponding dbspaces on the primary database server. You must perform a full restore.

Figure 20-7
Scenarios for Media Failure on the Secondary Database Server

HDR Server	Critical Media	Chunks Mirrored	Effect of Failure
Secondary	Yes	No	Secondary database server fails. Primary database server receives errors. HDR is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 20-27
Secondary	Yes	Yes	Secondary database server remains online in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 18-11 .
Secondary	No	No	Secondary database server remains online in read-only mode. Primary database server receives errors. HDR is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 20-27 .
Secondary	No	Yes	Secondary database server remains online in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 18-11 .

Restarting HDR After a Failure

[“HDR Failures Defined” on page 19-17](#) discusses the various types of HDR failures. The procedure that you must follow to restart HDR depends on whether or not critical data was damaged on one of the database servers. Both cases are discussed in this section.

Restarting After Critical Data Is Damaged

If one of the database servers experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks, and you are starting HDR for the first time. Use the functioning database server with the intact disks as the database server with the data.

Critical Media Failure on the Primary Database Server

You might need to restart HDR after the primary database server suffers a critical media failure. [Figure 20-8](#) lists the commands required to perform this procedure.

To restart HDR after a critical media failure

1. If the original secondary database server was changed to a standard database server, bring this database server to quiescent mode and then use the **onmode -d** command to change the type back to secondary.
2. Restore the primary database server from the last dbspace backup.
3. Use the **onmode -d** command to set the type of the primary database server and to start HDR.

The **onmode -d** command starts a logical recovery of the primary database server from the logical-log files on the secondary database server disk. If logical recovery cannot complete because you backed up and freed logical-log files on the original secondary database server, HDR does not start until you perform step 4.

4. Apply the logical-log files from the secondary database server, which were backed up to tape, to the primary database server.

If this step is required, the primary database server sends a message prompting you to recover the logical-log files from tape. This message appears in the message log. When all the required logical-log files have been recovered from tape, any remaining logical-log files on the secondary disk are recovered.

Figure 20-8
Steps for Restarting HDR After a Critical Media Failure on the Primary Database Server

Step	On the Primary Database Server	On the Secondary Database Server
1.		onmode command onmode -s onmode -d secondary <i>prim_name</i>
2.	ON-Bar command onbar -r -p ontape command ontape -p	
3.	onmode command onmode -d primary <i>sec_name</i>	
4.	ontape command ontape -l	

Critical Media Failure on the Secondary Database Server

If the secondary database server suffers a critical media failure, you can follow the same steps listed under [“Starting HDR for the First Time” on page 20-9](#).

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the computers that are running database servers in a replication pair experience a failure that damages the root dbspace, the dbspaces that contain logical-log files or the physical log, you need to restart HDR.

To restart HDR after a critical media failure on both database servers

1. Restore the primary database server from the storage space and logical-log backup.
2. After you restore the primary database server, treat the other failed database server as if it had no data on the disks and you were starting HDR for the first time.
(See [“Starting HDR for the First Time” on page 20-9.](#)) Use the functioning database server with the intact disks as the database server with the data.

Restarting If Critical Data Is Not Damaged

If no damage occurred to critical data on either database server, the following four scenarios, each requiring different procedures for restarting HDR, are possible:

- A network failure occurs.
- The secondary database server fails.
- The primary database server fails, and the secondary database server is not changed to a standard database server.
- The primary database server fails, and the secondary database server is changed to a standard database server.

Restarting After a Network Failure

After a network failure, the primary database server is in online mode, and the secondary database server is in read-only mode. HDR is turned off on both database servers (state = off). When the connection is reestablished, you can restart HDR by issuing **onmode -d secondary primary_name** on the secondary database server. Restarting HDR might not be necessary because the primary database server attempts to reconnect every 10 seconds and displays a message regarding the inability to connect every 2 minutes. You do not have to use onmode restart the connection.

Restarting If the Secondary Database Server Fails

If you need to restart HDR after a failure of the secondary database server, complete the steps in [Figure 20-9](#). The steps assume that you have been backing up logical-log files on the primary database server as necessary since the failure of the secondary database server.

Figure 20-9

Steps in Restarting After a Failure on the Secondary Database Server

Step	On the Primary	On the Secondary
1.	The primary database server should be in online mode.	<code>oninit</code> If you receive the following message in the message log, continue with step 2: <code>DR: Start Failure recovery from tape</code>
2.		ontape command <code>ontape -l</code>

Restarting If the Primary Database Server Fails

The following sections describe how to restart HDR if the primary database server fails under various circumstances.

The Secondary Database Server Was Not Changed to a Standard Database Server

If you need to restart HDR after a failure of the primary database server if the secondary database server is not changed to standard, simply bring the primary database server back online using **oninit**.

The Secondary Database Server Is Changed to a Standard Database Server

If you need to restart HDR after a failure of the primary database server, and you have changed the secondary database server to be a standard database server, complete the steps in [Figure 20-10](#).

Figure 20-10
Steps to Restart If You Changed the Secondary Database Server to Standard

Step	On the Primary Database Server	On the Secondary Database Server
1.		<p><code>onmode -s</code></p> <p>This step takes the secondary database server (now standard) to quiescent mode. All clients that are connected to this database server must disconnect. Applications that perform updates must be redirected to the primary. See “Redirection and Connectivity for Data-Replication Clients” on page 19-21.</p>
2.		<code>onmode -d secondary prim_name</code>
3.	<p><code>oninit</code></p> <p>If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the oninit command.</p> <p>If you have backed up and freed the logical-log files on the secondary, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 4).</p> <p>For ontape users:</p> <p>If you want to read the logical-log records over the network, set the logical-log tape device to a device on the computer that is running the secondary database server.</p>	
4.	<p>If you are prompted to recover logical-log records from tape, perform this step.</p> <p>ontape command</p> <p><code>ontape -l</code></p>	

Consistency Checking

In This Chapter	21-3
Performing Periodic Consistency Checking	21-4
Verifying Consistency	21-4
Validating System Catalog Tables	21-5
Validating Data Pages	21-5
Validating Extents	21-6
Validating Indexes	21-6
Validating Logical Logs	21-6
Validating Reserved Pages	21-6
Validating Metadata	21-6
Monitoring for Data Inconsistency	21-7
Reading Assertion Failures in the Message Log and Dump Files	21-7
Validating Table and Tablespace Data	21-8
Retaining Consistent Level-0 Backups	21-9
Dealing with Corruption	21-9
Finding Symptoms of Corruption	21-9
Fixing Index Corruption	21-10
Fixing I/O Errors on a Chunk	21-10
Collecting Diagnostic Information	21-11
Disabling I/O Errors	21-12
Monitoring the Database Server for Disabling I/O Errors	21-13
Using the Message Log to Monitor Disabling I/O Errors	21-13
Using Event Alarms to Monitor Disabling I/O Errors	21-14
Using No Bad-Sector Mapping	21-15

In This Chapter

Informix database servers are designed to detect database server malfunctions or problems caused by hardware or operating-system errors. It detects problems by performing *assertions* in many of its critical functions. An *assertion* is a consistency check that verifies that the contents of a page, structure, or other entity match what would otherwise be assumed.

When one of these checks finds that the contents are not what they should be, the database server reports an *assertion failure* and writes text that describes the check that failed in the database server message log. The database server also collects further diagnostics information in a separate file that might be useful to IBM Informix Technical Support staff.

This chapter provides an overview of consistency-checking measures and ways of handling inconsistencies. It covers the following topics:

- Performing periodic consistency checking
- Dealing with data corruption
- Collecting advanced diagnostic information
- Monitoring the database server for disabling I/O errors

Performing Periodic Consistency Checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, it is recommended that you periodically take the following actions:

- Verify that all data and the database server overhead information is consistent.
- Check the message log for assertion failures while you verify consistency.
- Create a level-0 dbspace backup after you verify consistency.

The following sections describe each of these actions.

Verifying Consistency

Because of the time needed for this check and the possible contention that the check can cause, schedule this check for times when activity is at its lowest. It is recommended that you perform this check just before you create a level-0 backup.

Run the commands shown in [Figure 21-1](#) as part of the consistency check.

Figure 21-1
Checking Data Consistency

Type of Validation	Command
System catalog tables	<code>oncheck -cc</code>
Data	<code>oncheck -cD dbname</code>
Extents	<code>oncheck -ce</code>
Indexes	<code>oncheck -cI dbname</code>
Reserved pages	<code>oncheck -cr</code>
Logical logs and reserved pages	<code>oncheck -cR</code>
Metadata and smart large objects	<code>oncheck -cs</code>

You can run each of these commands while the database server is in online mode. For information about how *each command* locks objects as it checks them and which users can perform validations, see **oncheck** in the *Administrator's Reference*.

In most cases, if one or more of these validation procedures detects an error, the solution is to restore the database from a dbspace backup. However, the source of the error might also be your hardware or operating system.

Validating System Catalog Tables

To validate system catalog tables, use the **oncheck -cc** command.

Each database contains its own system catalog, which contains information about the database tables, columns, indexes, views, constraints, stored procedures, and privileges.

If a warning appears when validation completes, its only purpose is to alert you that no records of a specific type were found. These warnings do not indicate any problem with your data, your system catalog, or even your database design. This warning indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**. For example, the following warning might appear if you validate system catalog tables for a database that has no synonyms defined for any table:

```
WARNING: No syssyntable records found.
```

However, if you receive an error message when you validate system catalog tables, the situation is quite different. Contact IBM Informix Technical Support immediately.

Validating Data Pages

To validate data pages, use the **oncheck -cD** command.

If data-page validation detects errors, try to unload the data from the specified table, drop the table, re-create the table, and reload the data. For information about loading and unloading data, see the *IBM Informix Migration Guide*. If this procedure does not succeed, perform a data restore from a storage-space backup.

Validating Extents

To validate extents in every database, use the **oncheck -ce** command.

Extents must not overlap. If this command detects errors, perform a data restore from a storage-space backup.

Validating Indexes

To validate indexes on each of the tables in the database, use the **oncheck -cI** command.

If this command detects errors, drop and re-create the affected index.

Validating Logical Logs

To validate logical logs and the reserved pages, use the **oncheck -cR** command.

Validating Reserved Pages

To validate reserved pages, use the **oncheck -cr** command.

Reserved pages are pages that reside at the beginning of the initial chunk of the root dbspace. These pages contain the primary database server overhead information. If this command detects errors, perform a data restore from storage-space backup.

This command might provide warnings. In most cases, these warnings call your attention to situations of which you are already aware.

Validating Metadata

Execute **oncheck -cs** for each database to validate metadata for all smart large objects in a database. If necessary, restore the data from a dbspace backup.

Monitoring for Data Inconsistency

If the consistency-checking code detects an inconsistency during database server operation, an assertion failure is reported to the database server message log. (See the message-log chapter in the *Administrator's Reference*.)

Reading Assertion Failures in the Message Log and Dump Files

Figure 21-2 shows the form that assertion failures take in the message log.

```
Assert Failed: Short description of what failed
Who: Description of user/session/thread running at the time
Result: State of the affected database server entity
Action: What action the database server administrator should take
See Also: file(s) containing additional diagnostics
```

Figure 21-2
Form of Assertion Failures in the Message Log

UNIX

The See Also: line contains one or more of the following filenames:

- **af.xxx**
- **shmem.xxx**
- **gcore.xxx**
- **gcore.xxx**
- **/pathname/core ♦**

In all cases, **xxx** is a hexadecimal number common to all files associated with the assertion failures of a single thread. The files **af.xxx**, **shmem.xxx**, and **gcore.xxx** are in the directory that the ONCONFIG parameter DUMPDIR specifies.

The file **af.xxx** contains a copy of the assertion-failure message that was sent to the message log, as well as the contents of the current, relevant structures and data buffers.

The file **shmem.xxx** contains a complete copy of the database server shared memory at the time of the assertion failure, but only if the ONCONFIG parameter DUMPSHMEM is set to 1.

UNIX

On UNIX, **gcore.xxx** contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPGCORE is set to 1 and your operating system supports the **gcore** utility. The **core** file contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPCORE is set to 1. The pathname for the **core** file is the directory from which the database server was last invoked. ♦

Validating Table and Tablespace Data

To validate table and tablespace data, use the **oncheck -cD** command on the database or table.

Most of the general assertion-failure messages are followed by additional information that usually includes the tblspace where the error was detected. If this check verifies the inconsistency, unload the data from the table, drop the table, re-create the table, and reload the data. Otherwise, no other action is needed.

In many cases, the database server stops immediately when an assertion fails. However, when failures appear to be specific to a table or smaller entity, the database server continues to run.

When an assertion fails because of inconsistencies on a data page that the database server accesses on behalf of a user, an error is also sent to the application process. The SQL error depends on the operation in progress. However, the ISAM error is almost always either -105 or -172, as follows:

```
-105 ISAM error: bad isam file format  
-172 ISAM error: Unexpected internal error
```

For additional details about the objectives and contents of messages, see the chapter on message-log messages in the *Administrator's Reference*.

Retaining Consistent Level-0 Backups

After you perform the checks described in “Verify Consistency” on page 30-4 without errors, create a level-0 backup. Retain this storage-space backup and all subsequent logical-log backup tapes until you complete the next consistency check. It is recommended that you perform the consistency checks before every level-0 backup. If you do not, then at minimum keep all the tapes necessary to recover from the storage-space backup that was created immediately after the database server was verified to be consistent.

Dealing with Corruption

This section describes some of the symptoms of database server system corruption and actions that the database server or you, as administrator, can take to resolve the problems. Corruption in a database can occur as a consequence of hardware or operating-system problems, or from some unknown database server problems. Corruption can affect either data or database server overhead information.

Finding Symptoms of Corruption

The database server alerts the user and administrator to possible corruption in the following ways:

- Error messages reported to the application state that pages, tables, or databases cannot be found. One of the following errors is always returned to the application if an operation has failed because of an inconsistency in the underlying data or overhead information:
 - 105 ISAM error: bad isam file format
 - 172 ISAM error: Unexpected internal error
- Assertion-failure reports are written to the database server message log. They always indicate files that contain additional diagnostic information that can help you determine the source of the problem. See [“Verifying Consistency” on page 21-4](#).
- The **oncheck** utility returns errors

Fixing Index Corruption

At the first indication of corruption, run the **oncheck -cI** command to determine if corruption exists in the index.

If you check indexes while the database server is in online mode, **oncheck** detects the corruption but does not prompt you for repairs. If corruption exists, you can drop and re-create the indexes using SQL statements while you are in online mode (the database server locks the table and index). If you run **oncheck -cI** in quiescent mode and corruption is detected, you are prompted to confirm whether the utility should attempt to repair the corruption.

Fixing I/O Errors on a Chunk

If an I/O error occurs during the database server operation, the status of the chunk on which the error occurred changes to *down*. If a chunk is *down*, the **onstat -d** display shows the chunk status as PD- for a primary chunk and MD- for a mirrored chunk.

Figure 21-3 shows an example in which chunk 2 is down.

```
Dbspaces
address  number  flags  fchunk  nchunks  flags  owner  name
40c980   1          1      1        1        N   informix rootdbs
40c9c4   2          1      2        1        N   informix fstddb
  2 active, 8192 maximum

Chunks
address  chk/dbs offset  size  free  bpages  flags  pathname
40c224   1  1  0    20000  14001  PO-    /home/server/root_chunk
40c2bc   2  2  0    2000   1659  PD-    /home/server/fst_chunk
  2 active, 8192 maximum
```

Figure 21-3
onstat -d Output

Additionally, the message log lists a message with the location of the error and a suggested solution. The listed solution is a possible fix, but does not always correct the problem.

If the down chunk is mirrored, the database server continues to operate using the mirrored chunk. Use operating-system utilities to determine what is wrong with the down chunk and correct the problem. You must then direct the database server to restore mirrored chunk data.

For information about recovering a mirrored chunk, refer to [“Recovering a Mirrored Chunk” on page 18-11](#).

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates an abort. Otherwise, the database server can continue to operate but cannot write to or read from the down chunk or any other chunks in the dbspace of that chunk. You must take steps to determine why the I/O error occurred, correct the problem, and restore the dbspace from a backup.

If you take the database server to offline mode when a chunk is marked as down (D), you can reinitialize the database server, provided that the chunk marked as down does not contain critical data (logical-log files, the physical log, or the root dbspace).

Collecting Diagnostic Information

Several ONCONFIG parameters affect the way in which the database server collects diagnostic information. Because an assertion failure is generally an indication of an unforeseen problem, notify IBM Informix Technical Support whenever one occurs. The diagnostic information collected is intended for the use of IBM Informix technical staff. The contents and use of **af.xxx** files and shared core are not further documented.

To determine the cause of the problem that triggered the assertion failure, it is critically important that you not destroy diagnostic information until IBM Informix Technical Support indicates that you can do so. Send email with the **af.xxx** file to IBM Informix Technical Support at tmail@us.ibm.com. The **af.xxx** file often contains information that they need to resolve the problem.

Several ONCONFIG parameters direct the database server to preserve diagnostic information whenever an assertion failure is detected or whenever the database server enters into an abort sequence:

- DUMPDIR
- DUMPSHMEM
- DUMPCNT
- DUMPCORE
- DUMPGCORE ♦

For more information about the configuration parameters, see the *Administrator's Reference*.

You decide whether to set these parameters. Diagnostic output can consume a large amount of disk space. (The exact content depends on the environment variables set and your operating system.) The elements of the output could include a copy of shared memory and a core dump.



Tip: *A core dump is an image of a process in memory at the time that the assertion failed. On some systems, core dumps include a copy of shared memory. Core dumps are useful only if this is the case.*

Database server administrators with disk-space constraints might prefer to write a script that detects the presence of diagnostic output in a specified directory and sends the output to tape. This approach preserves the diagnostic information and minimizes the amount of disk space used.

Disabling I/O Errors

Informix divides disabling I/O errors into two general categories: destructive and nondestructive. A disabling I/O error is destructive when the disk that contains a database becomes damaged in some way. This type of event threatens the integrity of data, and the database server marks the chunk and dbspace as down. The database server prohibits access to the damaged disk until you repair or replace the disk and perform a physical and logical restore.

A disabling I/O error is nondestructive when the error does not threaten the integrity of your data. Nondestructive errors occur when someone accidentally disconnects a cable, you somehow erase the symbolic link that you set up to point to a chunk, or a disk controller becomes damaged.

Before the database server considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when the database server attempts to perform an operation on a chunk that has at least one of the following characteristics:

- The chunk has no mirror.
- The primary or mirror companion of the chunk under question is offline.

Second, the error must occur when the database server attempts unsuccessfully to perform one of the following operations:

- Seek, read, or write on a chunk
 - Open a chunk
 - Verify that chunk information on the first used page is valid
- The database server performs this verification as a sanity check immediately after it opens a chunk.

You can prevent the database server from marking a dbspace as down while you investigate disabling I/O errors. If you find that the problem is trivial, such as a loose cable, you can bring the database server offline and then online again without restoring the affected dbspace from backup. If you find that the problem is more serious, such as a damaged disk, you can use **onmode -O** to mark the affected dbspace as down and continue processing.

Monitoring the Database Server for Disabling I/O Errors

The database server notifies you about disabling I/O errors in two ways:

- Message log
- Event alarms

Using the Message Log to Monitor Disabling I/O Errors

The database server sends the following message to the message log when a disabling I/O error occurs:

```
Assert Failed: Chunk {chunk-number} is being taken OFFLINE.  
Who: Description of user/session/thread running at the time  
Result: State of the affected database server entity  
Action: What action the database server administrator should take  
See Also: DUMPDIR/af.uniqid containing more diagnostics
```

The result and action depend on the current setting of ONDBSPACEDOWN, as described in the following table.

ONDBSPACEDOWN Setting	Result	Action
0	Dbspace {space_name} is disabled.	Restore dbspace {space_name}.
	Blobspace {space_name} is disabled.	Restore blobspace {space_name}.
1	The database server must abort.	Shut down and restart the database server.
2	The database server blocks at next checkpoint.	Use onmode -k to shut down, or use onmode -O to override.

For more information about interpreting messages that the database server sends to the message log, see the chapter about message-log messages in the *Administrator's Reference*.

Using Event Alarms to Monitor Disabling I/O Errors

When a dbspace incurs a disabling I/O error, the database server passes the following values as parameters to your event-alarm executable file.

Parameter	Value
Severity	4 (Emergency)
Class	5
Class message	Dbspace is disabled: 'dbspace-name'
Specific message	Chunk {chunk-number} is being taken OFFLINE.

If you want the database server to use event alarms to notify you about disabling I/O errors, write a script that the database server executes when it detects a disabling I/O error. For information about how to set up this executable file that you write, see the appendix on event alarms and the chapter on configuration parameters in the *Administrator's Reference*.

Using No Bad-Sector Mapping

Dynamic Server relies on the operating system of your host computer for bad-sector mapping. The database server learns of a bad sector or a bad track when it receives a failure return code from a system call. When this situation occurs, the database server retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, the database server marks as *down* the chunk where the read or write was attempted.

The database server cannot take any action to identify the bad cylinder, track, or sector location because the only information available is the byte displacement within the chunk where the I/O operation was attempted.

If the database server detects an I/O error on a chunk that is *not* mirrored, it marks the chunk as down. If the down chunk contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates an abort. Otherwise, the database server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

Distributed Data

Chapter 22 Multiphase Commit Protocols

Chapter 23 Recovering Manually from Failed Two-Phase Commit

Section V



Multiphase Commit Protocols

In This Chapter	22-3
Transaction Managers	22-3
Using the TP/XA Library With a Transaction Manager	22-4
Using Microsoft Transaction Server (MTS/XA)	22-4
Using Loosely-Coupled and Tightly-Coupled Modes.	22-5
Two-Phase Commit Protocol	22-5
When the Two-Phase Commit Protocol Is Used.	22-6
Two-Phase Commit Concepts	22-7
Phases of the Two-Phase Commit Protocol	22-8
Precommit Phase	22-8
Postdecision Phase	22-9
How the Two-Phase Commit Protocol Handles Failures.	22-9
Types of Failures That Automatic Recovery Handles	22-9
Administrator's Role in Automatic Recovery	22-10
Automatic-Recovery Mechanisms for Coordinator Failure	22-10
Automatic-Recovery Mechanisms for Participant Failure	22-10
Presumed-Abort Optimization	22-10
Independent Actions	22-11
Situations That Initiate Independent Action	22-11
Possible Results of Independent Action	22-12
Independent Actions That Allow Transactions to Complete Successfully	22-12
Independent Actions That Result in an Error Condition	22-13
Independent Actions That Result in Heuristic Decisions	22-14
The Heuristic Rollback Scenario	22-15
Conditions That Result in a Heuristic Rollback	22-15
Results of a Heuristic Rollback	22-17
The Heuristic End-Transaction Scenario	22-19

When to Perform a Heuristic End Transaction	22-19
How to Use onmode -Z.	22-21
Action When the Transaction Is Ended Heuristically.	22-21
Monitoring a Global Transaction	22-22
Two-Phase Commit Protocol Errors	22-23
Two-Phase Commit and Logical-Log Records.	22-23
Logical-Log Records When the Transaction Commits.	22-25
Logical-Log Records Written During a Heuristic Rollback	22-27
Logical-Log Records Written After a Heuristic End Transaction	22-29
Configuration Parameters Used in Two-Phase Commits	22-31
Function of the DEADLOCK_TIMEOUT Parameter	22-31
Function of the TXTIMEOUT Parameter	22-31
Heterogeneous Commit Protocol	22-32
Gateways That Can Participate in a Heterogeneous Commit Transaction	22-33
Enabling and Disabling of Heterogeneous Commit	22-34
How Heterogeneous Commit Works	22-34
Precommit Phase	22-35
Gateway Commit Phase	22-35
Heterogeneous Commit Optimization.	22-36
Implications of a Failed Heterogeneous Commit	22-36
Database Server Coordinator Failure	22-36
Participant Failure	22-37
Interpretation of Heterogeneous Commit Error Messages	22-38

In This Chapter

A *two-phase commit protocol* ensures that transactions are uniformly committed or rolled back across multiple database servers. You can use Informix database servers with IBM Informix Enterprise Gateway products or transaction managers to manipulate data in non-Informix databases. Distributed queries across Informix database servers support two-phase commit.

The *heterogeneous commit protocol* ensures that updates to one or more Informix databases and *one* non-Informix database in a single transaction are uniformly committed or rolled back.

For information on recovering manually from a failed two-phase commit, see [Chapter 23, “Recovering Manually from Failed Two-Phase Commit.”](#)

Transaction Managers

Transaction managers support two-phase commit and roll back. For example, if your database is Informix, your accounting system is Oracle, and your remittance system is Sybase, you can use a transaction manager to communicate between the different databases. You also can use transaction managers to ensure data consistency between Informix or non-Informix databases by using distributed transactions instead of Enterprise Replication or High-Availability Data Replication.

Using the TP/XA Library With a Transaction Manager

A *global transaction* is a distributed query where more than one database server is involved in the query. A global transaction environment has the following parts:

- The client application
- The resource manager (Informix database server)
- The transaction manager (third-party software)

TP/XA is a library of functions that lets the database server act as a resource manager in the X/Open DTP environment. Install the TP/XA library as part of IBM Informix ESQL/C to enable communication between a third-party transaction manager and the database server. The X/Open environment supports large-scale, high-performance OLTP applications. For more information, see the *TP/XA Programmer's Manual*.

Use TP/XA when your database has the following characteristics:

- Data is distributed across multivendor databases
- Transactions include Informix and non-Informix data

Using Microsoft Transaction Server (MTS/XA)

The database server supports the Microsoft Transaction Server (MTS/XA) as a transaction manager in the XA environment. To use MTS/XA, install IBM Informix Client Software Developer's Kit, the latest version of IBM Informix ODBC Driver, and MTS/XA. MTS/XA works on Windows. For more information, contact IBM Informix Technical Support, and see the *IBM Informix Client Products Installation Guide* and the MTS/XA documentation.

Using Loosely-Coupled and Tightly-Coupled Modes

The database server supports XA global transactions in loosely-coupled and tightly-coupled modes. The Tuxedo transaction manager, provided by BEA systems, supports loosely-coupled mode. Tuxedo works on both UNIX and Windows. The MTS/XA transaction manager uses tightly-coupled mode. For a complete list of supported transaction managers, contact your sales representative.

Loosely coupled mode means that the different database servers coordinate transactions, but do not share resources. The records from all branches of the transactions display as separate transactions in the logical log.

Tightly coupled mode means that the different database servers coordinate transactions and share resources such as locking and logging. The records from all branches of the transactions display as a single transaction in the logical log.

Two-Phase Commit Protocol

The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction. The two-phase commit protocol ensures that all participating database servers receive and implement the same action (either to commit or to roll back a transaction), regardless of local or network failure.

If any database server is unable to commit its portion of the transaction, *all* database servers participating in the transaction must be prevented from committing their work.

When the Two-Phase Commit Protocol Is Used

A database server automatically uses the two-phase commit protocol for any transaction that modifies data on multiple database servers.

For example, consider the configuration shown in [Figure 22-1](#).

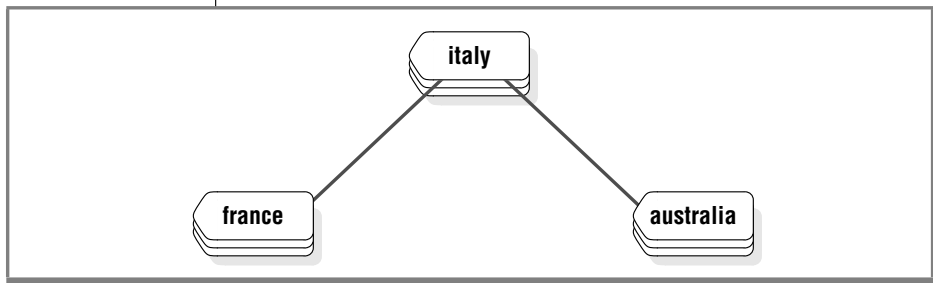


Figure 22-1
*Connected
Database Servers*

If you execute the commands shown in [Figure 22-2](#), the result is one update and two inserts at three different database servers.

```
CONNECT TO stores_demo@italy
BEGIN WORK
  UPDATE stores_demo:manufact SET manu_code = 'SHM' WHERE manu_name =
'Shimara'
  INSERT INTO stores_demo@france:manufact VALUES ('SHM', 'Shimara', '30')
  INSERT INTO stores_demo@australia:manufact VALUES ('SHM', 'Shimara', '30')
COMMIT WORK
```

Figure 22-2
*Example of a
Distributed
Transaction*

Two-Phase Commit Concepts

Every global transaction has a *coordinator* and one or more *participants*, defined as follows:

- The *coordinator* directs the resolution of the global transaction. It decides whether the global transaction should be committed or aborted.

The two-phase commit protocol always assigns the role of coordinator to the *current* database server. The role of coordinator cannot change during a single transaction. In the sample transaction in [Figure 22-2 on page 22-6](#), the coordinator is **italy**. If you change the first line in this example to the following statement, the two-phase commit protocol assigns the role of coordinator to **france**:

```
CONNECT TO stores_demo@france
```

Use the **onstat -x** option to display the coordinator for a distributed transaction. For more information, see [“Monitoring a Global Transaction” on page 22-22](#).

- Each *participant* directs the execution of one *transaction branch*, which is the part of the global transaction involving a single local database. A global transaction includes several transaction branches when:
 - An application uses multiple processes to work for a global transaction
 - Multiple remote applications work for the same global transaction

In [Figure 22-1 on page 22-6](#), the participants are **france** and **australia**. The coordinator database server, **italy**, also functions as a participant because it is also doing an update.

The two-phase commit protocol relies on two kinds of communication, *messages* and *logical-log records*:

- Messages pass between the coordinator and each participant. Messages from the coordinator include a transaction identification number and instructions (such as `prepare to commit`, `commit`, or `roll back`). Messages from each participant include the transaction status and reports of action taken (such as `can commit` or `cannot commit`, `committed`, or `rolled back`).
- Logical-log records of the transaction are kept on disk or tape to ensure data integrity and consistency, even if a failure occurs at a participating database server (participant or coordinator).
For more details, refer to [“Two-Phase Commit and Logical-Log Records” on page 22-23](#).

Phases of the Two-Phase Commit Protocol

In a two-phase commit transaction, the coordinator sends all the data modification instructions (for example, inserts) to all the participants. Then, the coordinator starts the two-phase commit protocol. The two-phase commit protocol has two parts, the *precommit phase* and the *postdecision phase*.

Precommit Phase

During the precommit phase, the coordinator and participants perform the following dialog:

1. **Coordinator.** The coordinator directs each participant database server to prepare to commit the transaction.
2. **Participants.** Every participant notifies the coordinator whether it can commit its transaction branch.
3. **Coordinator.** The coordinator, based on the response from each participant, decides whether to commit or roll back the transaction. It decides to commit only if *all* participants indicate that they can commit their transaction branches. If any participant indicates that it is *not* ready to commit its transaction branch (or if it does not respond), the coordinator decides to abort the global transaction.

Postdecision Phase

During the postdecision phase, the coordinator and participants perform the following dialog:

1. **Coordinator:** The coordinator writes the commit record or rollback record to the coordinator's logical log and then directs each participant database server to either commit or roll back the transaction.
2. **Participants:** If the coordinator issued a commit message, the participants commit the transaction by writing the commit record to the logical log and then sending a message to the coordinator acknowledging that the transaction was committed. If the coordinator issued a rollback message, the participants roll back the transaction but do not send an acknowledgment to the coordinator.
3. **Coordinator:** If the coordinator issued a message to commit the transaction, it waits to receive acknowledgment from each participant before it ends the global transaction. If the coordinator issued a message to roll back the transaction, it does not wait for acknowledgments from the participants.

How the Two-Phase Commit Protocol Handles Failures

The two-phase commit protocol is designed to handle system and media failures in such a way that data integrity is preserved across all the participating database servers. The two-phase commit protocol performs an *automatic recovery* if a failure occurs.

Types of Failures That Automatic Recovery Handles

The following events can cause the coordinating thread or the participant thread to terminate or hang, thereby requiring automatic recovery:

- System failure of the coordinator
- System failure of a participant
- Network failure
- Termination of the coordinating thread by the administrator
- Termination of the participant thread by the administrator



Administrator's Role in Automatic Recovery

The only role of the administrator in automatic recovery is to bring the coordinator or participant (or both) back online after a system or network failure.

Important: *A slow network cannot, and should not, trigger automatic recovery. None of the recovery mechanisms described here go into effect unless a coordinator system fails, a network fails, or the administrator terminates the coordinating thread.*

Automatic-Recovery Mechanisms for Coordinator Failure

If the coordinating thread fails, each participant database server must decide whether to initiate automatic recovery *before* it commits or rolls back the transaction or *after* it rolls back a transaction. This responsibility is part of the presumed-abort optimization. (See [“Presumed-Abort Optimization” on page 22-10.](#))

Automatic-Recovery Mechanisms for Participant Failure

Participant recovery occurs whenever a participant thread precommits a piece of work that is terminated before the two-phase commit protocol can be completed. The goal of participant recovery is to complete the two-phase commit protocol according to the decision reached by the coordinator.

Participant recovery is driven by either the coordinator or the participant, depending on whether the coordinator decided to commit or to roll back the global transaction.

Presumed-Abort Optimization

Presumed-abort optimization is the term that describes how the two-phase commit protocol handles the rollback of a transaction (an abort).

Rollback is handled in the following manner. When the coordinator determines that the transaction must be rolled back, it sends a message to all the participants to roll back their piece of work. The coordinator does not wait for an acknowledgment of this message, but proceeds to close the transaction and remove it from shared memory. If a participant tries to determine the status of this transaction—that is, find out whether the transaction was committed or rolled back (during participant recovery, for example)—it does not find any transaction status in shared memory. The participant must interpret this as meaning that the transaction was rolled back.

Independent Actions

An independent action in the context of two-phase commit is an action that occurs independently of the two-phase commit protocol. Independent actions might or might not be in opposition to the actions that the two-phase commit protocol specifies. If the action *is* in opposition to the two-phase commit protocol, the action results in an error or a *heuristic decision*. Heuristic decisions can result in an inconsistent database and require manual two-phase commit recovery. Manual recovery is an extremely complicated administrative procedure that you should try to avoid. (For a discussion of the manual-recovery process, see [Chapter 23, “Recovering Manually from Failed Two-Phase Commit.”](#))

Situations That Initiate Independent Action

Independent action during a two-phase commit protocol is rare, but it can occur in the following situations:

- The participant’s piece of work develops into a long-transaction error and is rolled back.
- An administrator kills a participant thread during the postdecision phase of the protocol with **onmode -z**.

- An administrator kills a participant transaction (piece of work) during the postdecision phase of the protocol with **onmode -Z**.
- An administrator kills a global transaction at the coordinator database server with **onmode -z** or **onmode -Z** *after* the coordinator issued a commit decision *and* became aware of a participant failure. This action always results in an error, specifically error -716.

Possible Results of Independent Action

As mentioned earlier, not all independent actions are in opposition to the two-phase commit protocol. Independent actions can yield the following three possible results:

- Successful completion of the two-phase commit protocol
- An error condition
- A heuristic decision

If the action is not in opposition to the two-phase protocol, the transaction should either commit or roll back normally. If the action ends the global transaction prematurely, an error condition results. Ending the global transaction at the coordinator is not considered a heuristic decision. If the action is in opposition to the two-phase commit protocol, a heuristic decision results. All these situations are discussed in the sections that follow.

Independent Actions That Allow Transactions to Complete Successfully

Independent actions are not necessarily in opposition to the two-phase commit protocol. For example, if a piece of work at a participant database server is rolled back because it developed into a long transaction, *and* the coordinator issues a decision to roll back the global transaction, the database remains consistent.

Independent Actions That Result in an Error Condition

If you, as administrator at the coordinator database server, execute either **onmode -z** (kill the coordinator thread) or **onmode -Z** (kill the global transaction) after the coordinator issues its final *commit* decision, you are removing all knowledge of the transaction from shared memory at the coordinator database server.

This action is not considered a heuristic decision because it does not interfere with the two-phase protocol; it is either acceptable, or it interferes with participant recovery and causes an error.

The action is acceptable any time that all participants are able to commit the transaction without difficulty. In this case, your action to end the transaction forcibly is superfluous. The indication that you executed **onmode -Z** reaches the coordinator only when the coordinator is preparing to terminate the transaction.

In practice, however, you would probably consider executing **onmode -z** or **onmode -Z** at the coordinator database server only if you were attempting to hasten the conclusion of a global transaction that has remained open for an unusually long period. In this scenario, the source of the problem is probably a failure at some participant database server. The coordinator has not received acknowledgment that the participant committed its piece of work, and the coordinator is attempting to establish communication with the participant to investigate.

If you execute either **onmode -z** or **onmode -Z** while the coordinator is actively trying to reestablish communication, the coordinating thread obeys your instruction to die, but not before it writes error -716 into the database server message log. The action is considered an error because the two-phase commit protocol was forcibly broken, preventing the coordinator from determining whether the database is consistent.

Killing a global transaction at a coordinator database server is not considered a heuristic decision, but it can result in an inconsistent database. For example, if the participant eventually comes back online and does not find the global transaction in the coordinator shared memory, it rolls back its piece of work, thereby causing a database inconsistency.

Independent Actions That Result in Heuristic Decisions

Some independent actions can develop into heuristic decisions when *both* of the following conditions are true:

- The participant database server already sent a `can commit` message to the coordinator and then rolls back.
- The coordinator's decision is to commit the transaction.

When both conditions are true, the net result is a global transaction that is inconsistently implemented (committed by one or more database servers and rolled back by another). The database becomes inconsistent.

The following two heuristic decisions are possible:

- Heuristic rollback (described in [“The Heuristic Rollback Scenario” on page 22-15](#))
- Heuristic end transaction (described in [“The Heuristic End-Transaction Scenario” on page 22-19](#)).

Once a heuristic rollback or end transaction occurs, you might have to perform manual recovery, a complex and time-consuming process. You need to understand heuristic decisions fully in order to avoid them. Always be wary of executing **onmode -z** or **onmode -Z** within the context of two-phase commit.

The Heuristic Rollback Scenario

In a *heuristic rollback*, either the database server or the administrator rolls back a piece of work that has already sent a `can commit` message.

Conditions That Result in a Heuristic Rollback

The following two conditions can cause a heuristic rollback:

- The logical log fills to the point defined by the `LTXEHWM` configuration parameter. (See the chapter on configuration parameters in the *Administrator's Reference*.) The source of the long-transaction condition is a piece of work being performed on behalf of a global transaction.
- An administrator executes `onmode -z session_id` to kill a database server thread that is executing a piece of work being performed on behalf of a global transaction.

In either case, if the piece of work has already sent a `can commit` message to its coordinator, the action is considered a heuristic decision.

Condition 1: Logical Log Fills to a High-Watermark

Under two-phase commit, a participant database server that is waiting for instructions from the coordinator is blocked from completing its transaction. Because the transaction remains open, the logical-log files that contain records associated with this transaction cannot be freed. The result is that the logical log continues to fill because of the activity of concurrent users.

If the logical log fills to the value of the long-transaction high-watermark (`LTXHWM`) while the participant is waiting, the database server directs all database server threads that own long transactions to begin rolling them back. If a piece of work that is precommitted is the offending long transaction, the database server has initiated a heuristic rollback. That is, this database server is rolling back a precommitted piece of work without the instruction or knowledge of the coordinator.

Under two-phase commit, the logical-log files that contain records associated with the piece of work are considered open until an `ENDTRANS` logical-log record is written. This type of transaction differs from a transaction involving a single database server where a rollback actually closes the transaction.

The logical log might continue to fill until the exclusive high-watermark is reached (LTXEHWL). If this happens, all user threads are suspended except those that are currently rolling back or currently committing. In the two-phase commit scenario, the open transaction prevents you from backing up the logical-log files and freeing space in the logical log. Under these specific circumstances, the logical log can fill completely. If this happens, the participant database server shuts down, and you must perform a data restore.

Condition 2: System Administrator Executes onmode -z

You, as administrator, can decide to initiate a heuristic rollback of a precommitted piece of work by executing **onmode -z**. You might make this decision because you want to free the resources that are held by the piece of work. (If you kill the participant thread by executing **onmode -z**, you free all locks and shared-memory resources that are held by the participant thread even though you do not end the transaction.)

Results of a Heuristic Rollback

This section describes what happens at both the coordinator and participant when a heuristic rollback occurs and how this process can result in an inconsistent database:

1. At the participant database server where the rollback occurred, a record is placed in the database server logical log (type HEURTX). Locks and resources held by the transaction are freed. The participant thread writes the following message in the database server message log, indicating that a long-transaction condition and rollback occurred:


```
Transaction Completed Abnormally (rollback):
tx=address flags=0xnn
```
2. The coordinator issues postdecision phase instructions to commit the transaction.

The participant thread at the database server where the heuristic rollback occurred returns error message -699 to the coordinator as follows:

```
-699 Transaction heuristically rolled back.
```

This error message is not returned to the application at this point; it is an internal notification to the coordinator. The coordinator waits until all participants respond to the commit instruction. The coordinator does not determine database consistency until all participants report.
3. The next steps depend on the actions that occur at the other participants. Two possible situations are possible.

Situation 1: Coordinator Issues a Commit and All Participants Report Heuristic Rollbacks

The coordinator gathers all responses from participants. If every participant reports a heuristic rollback, the following events occur as a consequence:

1. The coordinator writes the following message to its own database-server message log:


```
Transaction heuristically rolled back.
```
2. The coordinator sends a message to all participants to end the transaction.

3. Each participant writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
4. The coordinator writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
5. The coordinator returns error -699 to the application, as follows:
`-699 Transaction heuristically rolled back.`
6. In this situation, all databases remain consistent.

Situation 2: Coordinator Issued a Commit; One Participant Commits and One Reports a Heuristic Rollback

The coordinator gathers all responses from participants. If at least one participant reports a heuristic rollback and at least one reports an acknowledgment of a commit, the result is referred to as a *mixed-transaction result*. The following events occur as a consequence:

1. The coordinator writes the following message to its own database server message log:
`Mixed transaction result. (pid=nn user=userid)`
The `pid` value is the user-process identification number of the coordinator process. The `user` value is the user ID associated with the coordinator process. Associated with this message are additional messages that list each of the participant database servers that reported a heuristic rollback. The additional messages take the following form:
`Participant database server dbservername heuristically rolled back.`
2. The coordinator sends a message to each participant that heuristically rolled back its piece of work, directing each one to end the transaction.
3. Each participant writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
4. The coordinator writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)

5. The coordinator returns error -698 to the application, as follows:
-698 Inconsistent transaction. Number and names of servers rolled back.
6. Associated with this error message is the list of participant database servers that reported a heuristic rollback. If a large number of database servers rolled back the transaction, this list could be truncated. The complete list is always included in the message log for the coordinator database server.

In this situation, examine the logical log at each participant database server site and determine whether your database system is consistent. (See [“Determining If a Transaction Was Implemented Inconsistently”](#) on page 23-4.)

The Heuristic End-Transaction Scenario

Heuristic end transaction is an independent action taken by the administrator to roll back a piece of work and remove all information about the transaction from the database server shared-memory transaction table. The heuristic end-transaction process is initiated when the administrator executes the **onmode -Z address** command.

Whenever you initiate a heuristic end transaction by executing **onmode -Z**, you remove critical information required by the database server to support the two-phase commit protocol and its automatic-recovery features. If you execute **onmode -Z**, it becomes your responsibility to determine whether your networked database system is consistent.

When to Perform a Heuristic End Transaction

You should execute the **onmode -Z** option to initiate a heuristic end transaction in only one, rare, situation. This situation occurs when a piece of work that has been heuristically rolled back remains open, preventing your logical-log files from becoming free. As a result, the logical log is dangerously close to full.

In general, the coordinator issues its commit-or-rollback decision within a reasonable period of time. However, if the coordinator fails and does not return online to end a transaction that was heuristically rolled back at your participant database server, you might face a serious problem.

The problem scenario begins in this way:

1. The participant thread that is executing a piece of work on behalf of a global transaction has sent a `can commit` response to the coordinator.
2. The piece of work waits for instructions from the coordinator.
3. While the piece of work is waiting, the logical log fills past the long-transaction high-watermark.
4. The piece of work that is waiting for instructions is the source of the long transaction. The participant database server directs the executing thread to roll back the piece of work. This action is a heuristic rollback.
5. The participant continues to wait for the coordinator to direct it to end the transaction. The transaction remains open. The logical log continues to fill.

If the coordinator contacts the participant and directs it to end the transaction in a reasonable period of time, no problem develops. The serious problem arises if the heuristic rollback occurs at a participant database server and subsequently the coordinator fails, preventing the coordinator from directing the participant to end the transaction.

As a consequence, the transaction remains open. The open transaction prevents you from backing up logical-log files and freeing space in the logical log. As the logical log continues to fill, it might reach the point specified by the exclusive-access, long-transaction high-watermark (LTXEHW). If this point is reached, normal processing is suspended. At some point after the high-watermark is reached, you must decide if the open transaction is endangering your logical log. The danger is that if the logical log fills completely, the database server shuts down, and you must perform a data restore.

You must decide whether to kill the transaction and protect your system against the possibility of filling the logical log, despite all the problems associated with executing **`onmode -Z`**, or to wait and see if communication with the coordinator can be reestablished in time to end the transaction before the logical log fills.

How to Use onmode -Z

The **onmode -Z address** command is intended for use only if communication between the coordinator and the participant is broken. To ensure that communication is really broken, the **onmode -Z** command does not execute unless the thread that was executing the piece of work has been dead for the amount of time specified by `TXTIMEOUT`. For more information on this option, see the utilities chapter of the *Administrator's Reference*.

The *address* parameter is obtained from **onstat -x output**. For more information on the **onstat -x** option, see the utilities chapter of the *Administrator's Reference*.

Action When the Transaction Is Ended Heuristically

When you execute **onmode -Z**, you direct the **onmode** utility to remove the participant transaction entry, which is located at the specified address, from the transaction table.

Two records are written in the logical log to document the action. The records are type `ROLLBACK` and `ENDTRANS`, or if the transaction was already heuristically rolled back, `ENDTRANS` only. The following message is written to the participant database server message log:

```
(time_stamp) Transaction Completed Abnormally (endtx): tx=address
flags:0xnn user username tty ttyid
```

The coordinator receives an error message from the participant where the **onmode -Z** occurred, in response to its `COMMIT` instruction. The coordinator queries the participant database server, which no longer has information about the transaction. The lack of a transaction-table entry at the participant database server indicates that the transaction committed. The coordinator assumes that the acknowledgment message was sent from the participant, but somehow it was not received. Because the coordinator *does not know* that this participant's piece of work did not commit, it does not generate messages indicating that the global transaction was inconsistently implemented. Only the administrator who executed the **onmode -Z** command is aware of the inconsistent implementation.

Monitoring a Global Transaction

Use the **onstat -x** command to track open transactions and determine whether they have been heuristically rolled back. In [Figure 22-3](#), the long transaction at address `a733748` was heuristically rolled back.

The **H** flag in the third position of the **flags** field means a heuristic rollback and the **G** flag in the fifth position means a global transaction. The flag in the second position indicates the mode that the transaction is running in (**L** flag indicates loosely-coupled mode and **T** flag indicates tightly-coupled mode).

The **curlog** and **logposit** fields provide the exact position of a logical-log record. If a transaction is not rolling back, **curlog** and **logposit** describe the position of the most recently written log record. When a transaction is rolling back, these fields describe the position of the most recently “undone” log record. This record is in log 20 at page offset 2 (the third page in the log) and byte offset `0x218`. As the transaction rolls back, the **curlog** and **logposit** values decrease. In a long transaction, the rate at which the **logposit** and **beginlg** values converge can help you estimate how much longer the rollback is going to take.

Figure 22-3
onstat -x Output

```
IBM Informix Dynamic Server      Version 9.40.U      -- Online (LONGTX)

Transactions
address  flags  userthread locks  beginlg curlog  logposit  isol  retrys
coord
a733018  A---- a701018    0      0      20      0x11a0   COMMIT 0
a7331e4  A---- a701638    0      0      0       0x0     COMMIT 0
a7333b0  A---- a701c58    0      0      0       0x0     COMMIT 0
a73357c  A---- a702278    0      0      0       0x0     COMMIT 0
a733748  --H-G 0          0      17     20      0x2218   COMMIT 0
a733ae0  A---- a7034d8    0      0      0       0x0     COMMIT 0
  6 active, 128 total, 10 maximum concurrent
```

You also can use the **onstat -u** and **onstat -k** commands to track transactions and the locks that they hold. For details, see the monitoring transactions section in your *Performance Guide*. For a description of the fields that **onstat -x** displays, see the utilities chapter of the *Administrator's Reference*.

Two-Phase Commit Protocol Errors

The following two-phase commit protocol errors require special attention from the administrator.

Error Number	Description
-698	If you receive error -698, a heuristic rollback has occurred and has caused an inconsistently implemented transaction. The circumstances leading up to this event are described in “Results of a Heuristic Rollback” on page 22-17 . For an explanation of how the inconsistent transaction developed and to learn the options available to you, refer to this discussion.
-699	If you receive error -699, a heuristic rollback has occurred. The circumstances leading up to this event are described in “Results of a Heuristic Rollback” on page 22-17 . For an explanation of how the inconsistent transaction developed, refer to this discussion.
-716	If you receive error -716, the coordinating thread has been terminated by administrator action after it issued its final decision. This scenario is described under “Independent Actions That Result in an Error Condition” on page 22-13 .

Two-Phase Commit and Logical-Log Records

The database server uses logical-log records to implement the two-phase commit protocol. You can use these logical-log records to detect heuristic decisions and, if necessary, to help you perform a manual recovery. (See [Chapter 23, “Recovering Manually from Failed Two-Phase Commit.”](#))

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

For information about these logical-log records, see the chapter on interpreting the logical log in the *Administrator's Reference*.

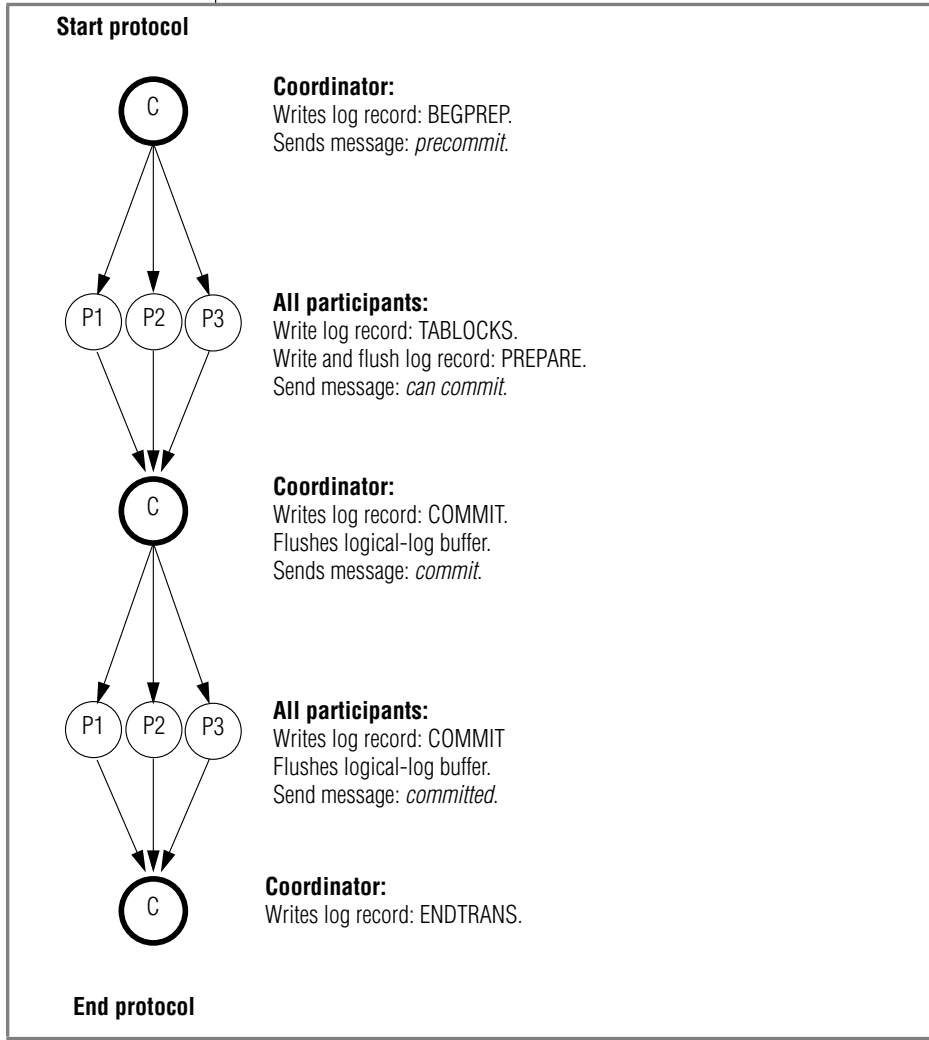
This section examines the sequence of logical-log records that are written during the following database server scenarios:

- A transaction is committed.
- A piece of work is heuristically rolled back.
- A piece of work is heuristically ended.

Logical-Log Records When the Transaction Commits

Figure 22-4 illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

Figure 22-4
Logical-Log
Records Written
During a Committed
Transaction



Some of the logical-log records must be flushed from the logical-log buffer immediately; for others, flushing is not critical.

The coordinator's commit-work record (COMMIT record) contains all information needed to initiate the two-phase commit protocol. It also serves as the starting point for automatic recovery in the event of a failure on the coordinator's host computer. Because this record is critical to recovery, it is not allowed to remain in the logical-log buffer. The coordinator must immediately flush the COMMIT logical-log record.

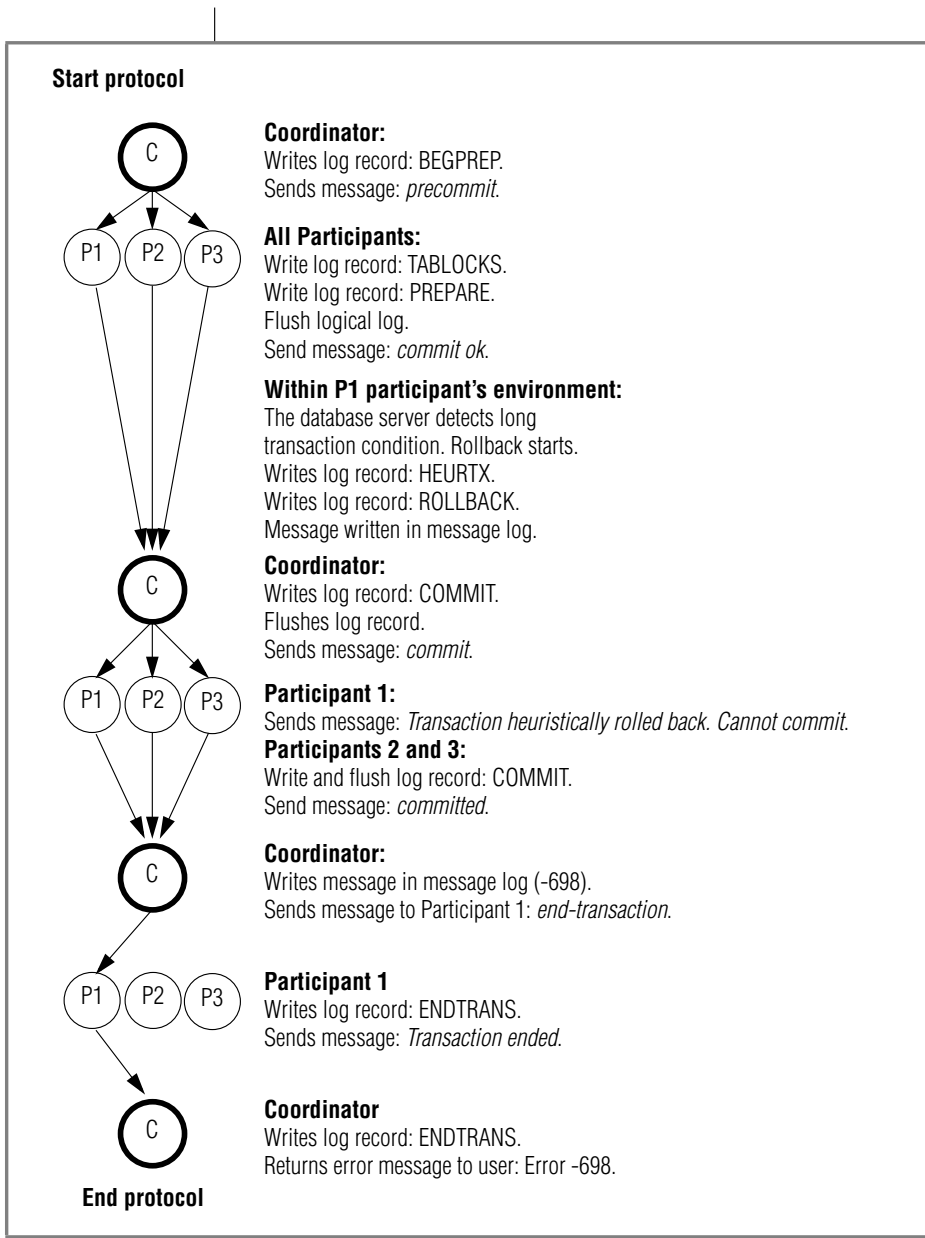
The participants in [Figure 22-4 on page 22-25](#) must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. As part of recovery, the participant might query the coordinator to learn the final disposition of this transaction.

Flushing the participant's COMMIT record ensures that, if the participant's host computer fails, the participant has a record of what action it took regarding the transaction. To understand why this information is crucial, consider the situation in which a participant crashes after the PREPARE record is written but before the COMMIT record flushes. After fast recovery, the PREPARE record is restored, but the COMMIT record is lost (because it was in the logical-log buffer at the time of the failure). The existence of the PREPARE record would initiate a query to the coordinator about the transaction. However, the coordinator would know nothing of the transaction, because it ended the transaction after it received the participant's acknowledgment that the commit occurred. In this situation, the participant would interpret the lack of information as a final direction to roll back the transaction. The two-phase commit protocol requires the participant's COMMIT record to be flushed immediately to prevent this kind of misunderstanding.

Logical-Log Records Written During a Heuristic Rollback

[Figure 22-5 on page 22-28](#) illustrates the sequence in which the database server writes the logical-log records during a heuristic rollback. Because a heuristic rollback only occurs after the participant sends a message that it can commit and the coordinator sends a message to commit, the first phase of this protocol is the same as that shown in [Figure 22-4 on page 22-25](#). When a heuristic rollback occurs, the rollback is assumed to be the consequence of a long-transaction condition that occurs at the Participant 1 (P1) database server. The end result is a transaction that is inconsistently implemented. See [“The Heuristic Rollback Scenario” on page 22-15](#).

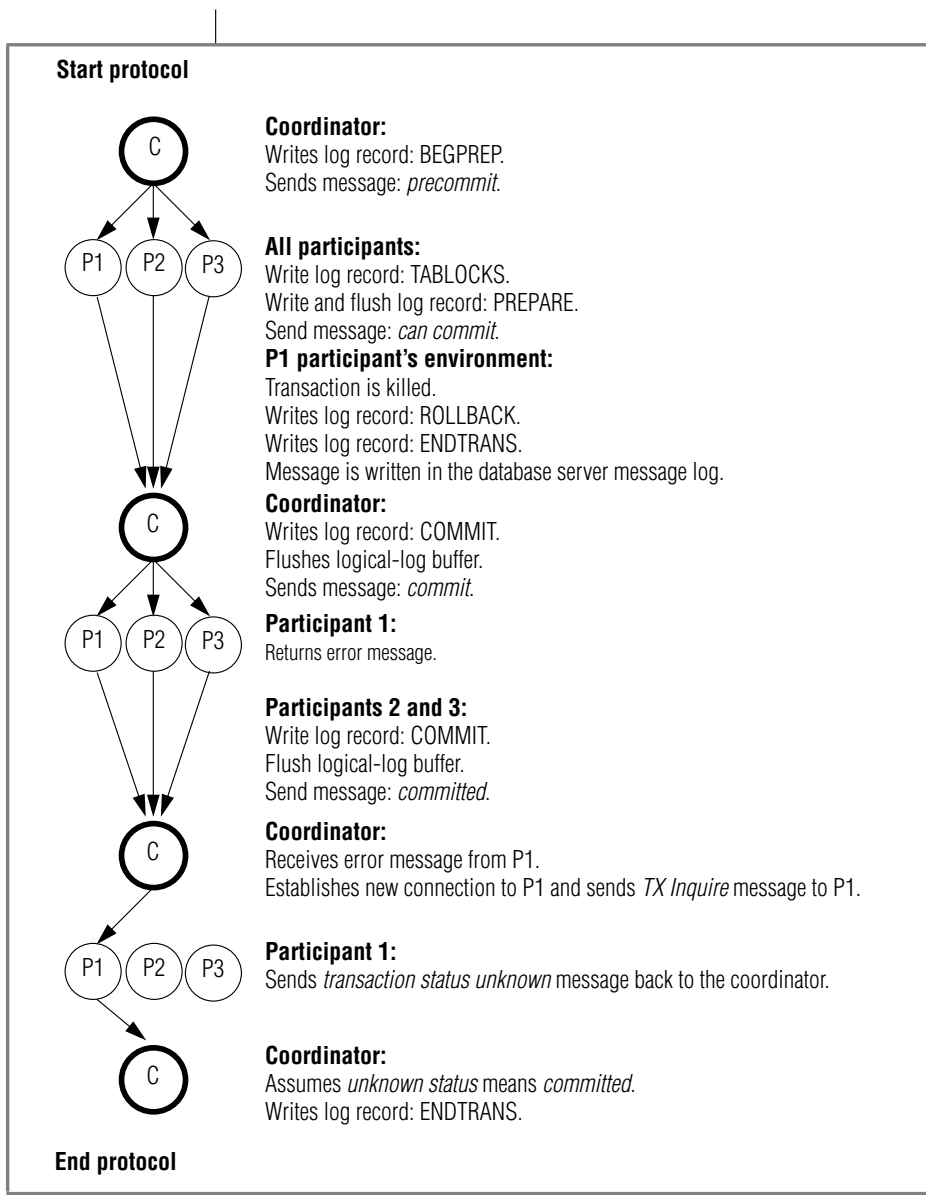
Figure 22-5
Logical-Log
Records Written
During a Heuristic
Rollback



Logical-Log Records Written After a Heuristic End Transaction

[Figure 22-6 on page 22-30](#) illustrates the writing sequence of the logical-log records during a heuristic end transaction. The event is always the result of a database server administrator killing a transaction (see the **onmode** section of the utilities chapter in the *Administrator's Reference*) at a participant database server after the participant has sent a `can commit` message. In [Figure 22-6](#), the heuristic end transaction is assumed to have occurred at the P1 participant. The result is an inconsistently implemented transaction. See [“The Heuristic End-Transaction Scenario” on page 22-19](#).

Figure 22-6
Logical-Log
Records Written
During a Heuristic
End Transaction



Configuration Parameters Used in Two-Phase Commits

The following two configuration-file parameters are specific to distributed environments:

- DEADLOCK_TIMEOUT
- TXTIMEOUT

Although both parameters specify time-out periods, the two are independent. For more information on these configuration parameters, see the *Administrator's Reference*.

Function of the DEADLOCK_TIMEOUT Parameter

If a distributed transaction is forced to wait longer than the number of seconds specified by DEADLOCK_TIMEOUT for a shared-memory resource, the thread that owns the transaction assumes that a multiserver deadlock exists. The following error message is returned:

```
-154 ISAM error: deadlock timeout expired - Possible deadlock.
```

The default value of DEADLOCK_TIMEOUT is 60 seconds. Adjust this value carefully. If you set it too low, individual database servers abort transactions that are not deadlocks. If you set it too high, multiserver deadlocks could reduce concurrency.

Function of the TXTIMEOUT Parameter

The TXTIMEOUT configuration parameter is specific to the two-phase commit protocol. It is used only if communication between a transaction coordinator and participant has been interrupted and needs to be reestablished.

The TXTIMEOUT parameter specifies a period of time that a participant database server waits to receive a *commit* instruction from a coordinator database server during a distributed transaction. If the period of time specified by TXTIMEOUT elapses, the participant database server checks the status of the transaction to determine if the participant should initiate automatic participant recovery.

TXTIMEOUT is specified in seconds. The default value is 300 (five minutes). The optimal value for this parameter varies, depending on your specific environment and application. Before you modify this parameter, read the discussion [“How the Two-Phase Commit Protocol Handles Failures”](#) on page 22-9.

Heterogeneous Commit Protocol

Used in the context of Informix database servers, the term heterogeneous environment refers to a group of database servers in which at least one of the database servers is not an Informix database server. Heterogeneous commit ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment.

Unlike the two-phase commit protocol, the heterogeneous commit protocol supports the participation of a non-Informix participant. The non-Informix participant, called a gateway participant, must communicate with the coordinator through an Informix gateway.

The database server uses heterogeneous commit protocol when the following criteria are met:

- Heterogeneous commit is enabled. (That is, the HETERO_COMMIT configuration parameter is set to 1.)
- The coordinator of the commit is a Version 7.2 or later IBM Informix Dynamic Server.
- The non-Informix participant communicates with the Informix database server through an Informix gateway.
- At most, one non-Informix participant performs an update within a single transaction.

Figure 22-7 illustrates this scenario.

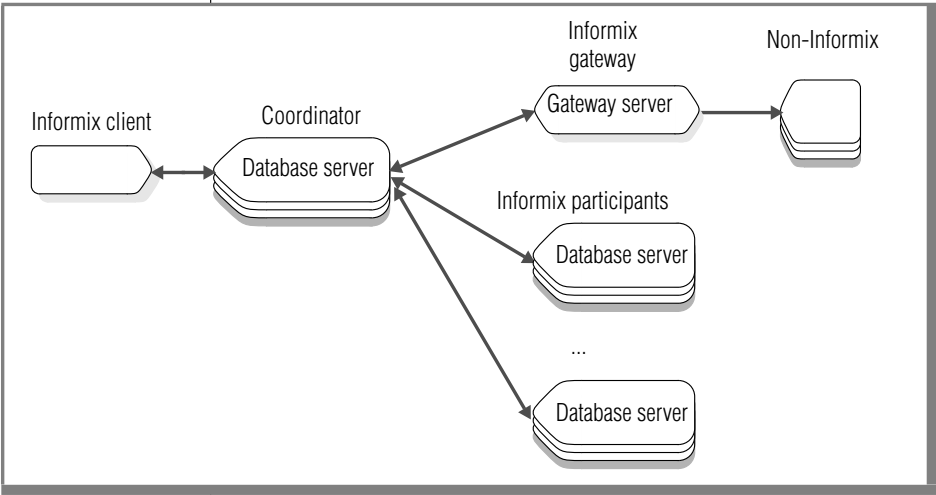


Figure 22-7
Configuration That
Requires
Heterogeneous
Commit for
Distributed
Transactions

Gateways That Can Participate in a Heterogeneous Commit Transaction

An Informix gateway acts as a bridge between an Informix application (in this case, a database server) and a non-Informix database server. An Informix gateway allows you to use an Informix application to access and modify data that is stored in non-Informix databases.

The following table lists the gateways and corresponding database servers that can participate in a transaction in which the database server uses the heterogeneous commit protocol.

Gateway	Database Servers
IBM Informix Enterprise Gateway with DRDA	IBM DB2, OS/400, SQL/DS
IBM Informix Enterprise Gateway for EDA/SQL	EDA/SQL
IBM Informix Enterprise Gateway Managerr	Any database server with ODBC connectivity

Enabling and Disabling of Heterogeneous Commit

Use a text editor or ISA to change the HETERO_COMMIT configuration parameter which enables or disables heterogeneous commit: The change takes effect when you shut down and restart the database server.

When you set HETERO_COMMIT to 1, the transaction coordinator checks for distributed transactions that require the use of heterogeneous commit. When the coordinator detects such a transaction, it automatically executes the heterogeneous commit protocol.

If you set HETERO_COMMIT to 0 or any number other than 1, the transaction coordinator disables the heterogeneous commit protocol. The following table summarizes which protocol the transaction coordinator uses, heterogeneous commit or two-phase commit, to ensure the integrity of a distributed transaction.

HETERO_COMMIT Setting	Gateway Participant Updated	Database Server Protocol
Disabled	No	Two-phase commit
Disabled	Yes	Two-phase commit
Enabled	No	Two-phase commit
Enabled	Yes	Heterogeneous commit

How Heterogeneous Commit Works

The heterogeneous commit protocol is a modified version of the standard two-phase commit protocol. The postdecision phase in the heterogeneous commit protocol is identical to the postdecision phases in the two-phase commit protocol. The precommit phase contains a minor modification, and a new phase, called the gateway commit phase, is added to the heterogeneous commit protocol.

The following sections explain the modification to the precommit phase and the gateway commit phase. For a detailed explanation of the postdecision phases, see [“Postdecision Phase” on page 22-9](#).

Precommit Phase

The coordinator directs each update participant (except the gateway participant) to prepare to commit the transaction.

If the updates satisfy all deferred constraints, all participants (except the gateway participant) return messages to the coordinator indicating that they can commit their piece of work.

Gateway Commit Phase

If all participants successfully return a message indicating that they are prepared to commit, the coordinator sends a commit message to the gateway. The gateway in turn sends a response to the coordinator indicating whether the gateway committed its piece of the transaction. If the gateway commits the transaction, the coordinator decides to commit the entire transaction.

[Figure 22-8 on page 22-35](#) illustrates this process.

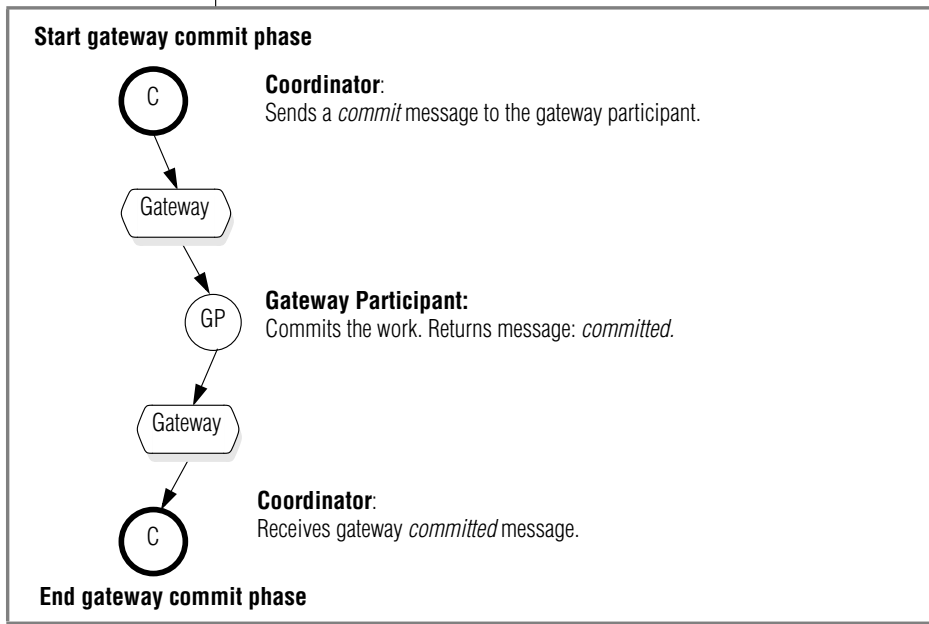


Figure 22-8
*Heterogeneous
Commit Phase
That Results in a
Committed
Transaction*

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as [Figure 22-8](#) illustrates.

Heterogeneous Commit Optimization

The database server optimizes the heterogeneous commit protocol when the only participant that receives an update is a non-Informix database. In this case, the coordinator sends a single commit message to all participants without invoking the heterogeneous commit protocol.

Implications of a Failed Heterogeneous Commit

At any time during a distributed transaction that the database server processes using heterogeneous commit, the coordinator or any number of participants can fail. The database server handles these failures in the same way as in the two-phase commit protocol, except in certain instances. The following sections examine these special instances in detail.

Database Server Coordinator Failure

The consistency of data after a coordinator failure depends on the point in the heterogeneous commit process at which the coordinator fails. If the coordinator fails before sending the commit message to the gateway, the entire transaction is aborted upon recovery, as is the case with two-phase commit.

If the coordinator fails after it writes the commit log record, the entire transaction is committed successfully upon recovery, as is the case with two-phase commit.

If the coordinator fails after it sends the commit message to the gateway but before it writes the commit log record, the remote Informix database server sites in the transaction are aborted upon recovery. This abort might result in inconsistencies if the gateway received the commit message and committed the transaction.

The following table summarizes these scenarios.

Point of Database Server Coordinator Failure	Expected Result
After the coordinator writes the PREPARE log record and before the gateway commit phase	Data consistency is maintained.
After the coordinator sends a commit message to the gateway but before it receives a reply	Data is probably inconsistent. No indication of probable data inconsistency from the coordinator.
After gateway commit phase but before the coordinator writes a COMMIT record to the logical log	Data consistency is lost. No indication of data inconsistency from the coordinator.

Participant Failure

Whenever a participant in a distributed transaction that uses the heterogeneous protocol fails, the coordinator sends the following error message:

-441 Possible inconsistent data at the target DBMS *name* due to an aborted commit.

In addition, the database server sends the following message to the message log:

Data source accessed using gateway *name* might be in an inconsistent state.

A participant failure is not limited to the failure of a database server or gateway. In addition, a failure of the communication link between the coordinator and the gateway is considered a gateway failure. The gateway terminates if a link failure occurs. The gateway must terminate because it does not maintain a transaction log and therefore cannot reestablish a connection with the coordinator and resume the transaction. Because of this restriction, some scenarios exist in which a gateway failure might leave data in an inconsistent state. The following table summarizes these scenarios.

Point of Participant Failure	Expected Result
After participant receives commit transaction message from coordinator, but before participant performs commit	Data consistency is maintained.
After participant receives commit transaction message from coordinator and commits the transaction, but before the participant replies to coordinator	Data is inconsistent.
After participant commits the transaction and sends a reply to coordinator	If the communications link fails before the coordinator receives the reply, then data is inconsistent. If the coordinator receives the reply, then data is consistent (provided the coordinator does not fail before writing the COMMIT record).

The recovery procedure that the database server follows when a participant fails is identical to the procedure that is followed in two-phase commit. For more information on this procedure, see [“Participant Failure” on page 22-37](#).

Interpretation of Heterogeneous Commit Error Messages

When the database server fails to process a distributed transaction using heterogeneous commit, it returns one of the two error messages that are discussed in the following sections.

Application Attempts to Update Multiple Gateway Participants

If your client application attempts to update data at more than one gateway participant when HETERO_COMMIT is set to 1, the coordinator returns the following error message:

```
-440 Cannot update more than one non-Informix DBMS within a
transaction.
```

If you receive this error message, rewrite the offending application so that it updates at most one gateway participant in a single distributed transaction.

Failed Attempt to Commit Distributed Transaction Using Heterogeneous Commit

The database server can fail to commit a distributed transaction while it is using the heterogeneous protocol for one or more of the following reasons:

- Communication error
- Site failure
- Gateway failure
- Other unknown error

When such a failure occurs, the coordinator returns the following message:

```
-441 Possible inconsistent data at the target DBMS name due to an
aborted commit.
```

After the database server sends this message, it rolls back all update sites that are participating in the transaction, with the possible exception of the work done at the site of the gateway participant. The gateway participant might have committed its updates if the failure occurred after the gateway participant processed the commit message. If the gateway participant committed the updates, you must manually rollback these updates.

Recovering Manually from Failed Two-Phase Commit

In This Chapter	23-3
Determining If Manual Recovery Is Required.	23-3
Determining If a Transaction Was Implemented Inconsistently	23-4
Global Transaction Killed Prematurely	23-4
Heuristic End Transaction.	23-4
Heuristic Rollback	23-5
Determining If the Distributed Database Contains	
Inconsistent Data	23-6
Obtaining Information from the Logical Log	23-7
Obtaining the Global Transaction Identifier.	23-8
Deciding If Action Is Needed to Correct the Situation	23-9
Example of Manual Recovery	23-10

In This Chapter

Distributed transactions follow the two-phase commit protocol. Certain actions occur independently of the two-phase commit protocol and cause the transaction to be inconsistently implemented. (See [“Independent Actions” on page 22-11.](#)) In these situations, it might be necessary to recover manually from the transaction.

This chapter describes the following topics:

- Determining if you need to recover manually from an inconsistently implemented two-phase commit transaction
- Performing a manual recovery

Determining If Manual Recovery Is Required

The following list outlines the steps in the procedure to determine database consistency and to correct the situation if required.

To determine database consistency

1. Determine whether a transaction was implemented inconsistently.
2. Determine if the networked database system contains inconsistent data.
3. Decide if action to correct the situation is required.

Each of these steps is described in the following sections.

Determining If a Transaction Was Implemented Inconsistently

Your first task is to determine whether the transaction was implemented inconsistently as a result of an independent action.

Global Transaction Killed Prematurely

If you executed an **onmode -z** command to kill the global transaction on the coordinator, the transaction might be inconsistently implemented. (For an explanation of how this situation can arise, see [“Independent Actions That Result in an Error Condition” on page 22-13.](#)) You can check for an inconsistent transaction by first examining the database server message log for the coordinator. Look for the following error message:

```
-716 Possible inconsistent transaction. Unknown servers are  
server-name-list.
```

This message lists all the database servers that were participants. Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic End Transaction

If you executed an **onmode -Z address** command to end a piece of work performed by a participant, *and* the coordinator decided to commit the transaction, the transaction is implemented inconsistently. (For a description of this scenario, see [“The Heuristic End-Transaction Scenario” on page 22-19.](#)) Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic Rollback

You can determine the specific database server participants affected by a heuristic decision to roll back a transaction in the following ways:

- Examine the return code from the COMMIT WORK statement in the application.

The following message indicates that one of the participants performed a heuristic rollback:

```
-698 Inconsistent transaction. Number and names of servers  
rolled back.
```

- Examine the messages in the database server message-log file.

If a database inconsistency is possible because of a heuristic decision at a participating database server, the following message appears in the database server message-log file of the coordinator:

```
Mixed transaction result. (pid=nn user=user_id)
```

This message is written whenever error -698 is returned. Associated with this message is a list of the participant database servers where the transaction was rolled back. This is the complete list. The list that appears with the -698 error message could be truncated if a large number of participants rolled back the transaction.

- Examine the logical log for each participant.

If at least one participant rolls back its piece of work and one participant commits its piece of work, the transaction is implemented incorrectly.

Determining If the Distributed Database Contains Inconsistent Data

If you determine that a transaction was inconsistently implemented, you must determine what this situation means for your distributed database system. Specifically, you must determine if data integrity has been affected.

A transaction that is inconsistently implemented causes problems whenever the piece of work rolled back by one participant is dependent on a piece of work that was updated by another participant. It is impossible to define these dependencies with SQL because distributed transactions do not support constraints that reference data at multiple database servers. The pieces of work are independent (no dependencies exist) only if the data could have been updated in two independent transactions. Otherwise, the pieces of work are considered to be dependent.

Before you proceed, consider the transaction that caused the error. Are the pieces of data that were updated and rolled back dependent on one another? Multiple updates might be included in a single transaction for reasons other than maintaining data integrity. For example, three possible reasons are as follows:

- Reduced transaction overhead
- Simplified coding
- Programmer preference

Verify also that every participant database server that is assumed to have committed the transaction actually modified data. A read-only database server might be listed as a participant that committed a transaction.

If an inconsistent transaction does not lead to a violation of data integrity, you can quit the procedure at this point.

Obtaining Information from the Logical Log

To determine if data integrity has been affected by an inconsistently implemented global transaction, you need to reconstruct the global transaction and determine which parts of the transaction were committed and which were rolled back. Use the **onlog** utility to obtain the necessary information. The procedure is as follows:

1. Reconstruct the transaction at the participant that contains the HEURTX record.
 - a. A participant database server logical log is the starting point for your information search. Each record in the log has a local transaction identification number (**xid**). Obtain the **xid** of the HEURTX record.
 - b. Use the local **xid** to locate all associated log records that rolled back as part of this piece of work.
2. Determine which database server acted as coordinator for the global transaction.
 - a. Look for the PREPARE record on the participant that contains the same local **xid**. The PREPARE record marks the start of the two-phase commit protocol for the participant.
 - b. Use the **onlog -l** option to obtain the long output of the PREPARE record. This record contains the global transaction identifier (GTRID) and the name of the coordinating database server. For information about GTRID, see [“Obtaining the Global Transaction Identifier” on page 23-8](#).
3. Obtain a list of the other participants from the coordinator log.
 - a. Examine the log records on the coordinator database server. Find the BEGPREP record.
 - b. Examine the long output for the BEGPREP record. If the first 32 bytes of the GTRID in this record match the GTRID of the participant, the BEGPREP record is part of the same global transaction. Note the participants displayed in the ASCII part of the BEGPREP long output.

4. Reconstruct the transaction at each participant.
 - a. At each participant database server, read the logical log to find the PREPARE record that contains the GTRID associated with this transaction and obtain the local **xid** for the piece of work performed by this participant.
 - b. At each participant database server, use the local **xid** to locate all logical-log records associated with this transaction (committed or rolled back).

After you follow this procedure, you know what all the participants for the transaction were, which pieces of work were assigned to each participant, and whether each piece of work was rolled back or committed. From this information, you can determine if the independent action affected data integrity.

Obtaining the Global Transaction Identifier

When a global transaction starts, it receives a unique identification number called a global transaction identifier (GTRID). The GTRID includes the name of the coordinator. The GTRID is written to the BEGPREP logical-log record of the coordinator and the PREPARE logical-log record of each participant.

To see the GTRID, use the **onlog -l** option. The GTRID is offset 20 bytes into the data portion of the record and is 144 bytes long. [Figure 23-1](#) shows the **onlog -l** output for a BEGPREP record. The coordinator is **chrisw**.

```
4a064 188 BEGPREP 4 0 4a038 0 1
000000bc 00000043 00000004 0004a038 .....C .....8
00087ef0 00000002 63687269 73770000 ..~.... chrisw..
00000000 00000000 00000000 00087eeb .....~.....
00006b16 00000000 00000000 00000000 ..k.....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000000 00000000 00000000 .....
00000000 00000001 6a756469 74685f73 ..... judith_s
6f630000 736f6374 63700000 oc..soct cp..
```

Figure 23-1
*Output of the
onlog -l Option for a
BEGPREP Record*

The first 32 bytes of the GTRID are identical for the BEGPREP record on the coordinator and the PREPARE records on participants, which are part of the same global transaction. For example, compare the GTRID for the PREPARE record in [Figure 23-2](#) with that of the BEGPREP record in [Figure 23-1](#).

```
c7064 184 PREPARE 4 0 c7038 chrism
000000b8 00000044 00000004 000c7038 .....D .....p8
00005cd6 00000002 63687269 73770000 ..... chrism..
00000000 00000000 00000069 00087eeb ..... .i.~.
00006b16 00000000 00000010 00ba5a10 ..k.....Z.
00000002 00ba3a0c 00000006 00000000 .....
00ba5a10 00ba5a1c 00000000 00000000 ..Z...Z.....
00ba3a0e 00254554 00ba2090 00000001 ....%ET ..
00000000 00ab8148 0005fd70 00ab8148 .....H ...p...H
0005fe34 0000003c 00000000 00000000 ...4...< .....
00000000 00ab80cc 00000000 00ab80c4 .....
00ba002f 63687269 73770000 00120018 .../chrism.....
00120018 00ba0000 .....
```

Figure 23-2
Output of the
onlog -l Option for a
PREPARE Record

Deciding If Action Is Needed to Correct the Situation

If an inconsistent transaction creates an inconsistent database, the following three options are available to you:

- Leave the networked database in its inconsistent state.
- Remove the effects of the transaction wherever it was committed, thereby rolling back the entire transaction.
- Reapply the effects of the transaction wherever it was rolled back, thereby committing the transaction.

You can leave the database in its inconsistent state if the transaction does not significantly affect database data. You might encounter this situation if the application that is performing the transaction can continue as it is, and you decide that the price (in time and effort) of returning the database to a consistent state by either removing the effects or reapplying the transaction is too high.

You do not have to reach this decision immediately. You can use the methods described in the following paragraphs to determine what data the transaction was updating and which records are affected.

As you make your decision, consider that no automatic process or utility can perform a rollback of a committed transaction or can commit part of a transaction that has been rolled back. The following paragraphs describe how to look through the database server message log and the logical log to locate affected records. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you must determine whether to roll back or to commit the transaction. You must also program the compensating transaction that will perform the rollback or the commit.

Example of Manual Recovery

This example illustrates the kind of work that is involved in manual recovery. The following SQL statements were executed by user **nhowe**. Error -698 was returned.

```
dbaccess
CREATE DATABASE tmp WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
CREATE DATABASE tmp@apex WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
DATABASE tmp;
BEGIN WORK;
INSERT INTO t VALUES (2);
INSERT INTO tmp@apex:t VALUES (2);
COMMIT WORK;
### return code -698
```

The following excerpt is taken from the logical log at the current database server:

addr	lentype	xid	id	link					
.....									
17018	16CKPOINT	0	0	13018	0				
18018	20BEGIN	2	1	0	08/27/91 10:56:57	3482		nhowe	
1802c	32HINSERT	2	0	18018	1000018	102	4		
1804c	40CKPOINT	0	0	17018	1				
begin		xid	id	addr	user				
1		2	1	1802c	nhowe				
19018	72BEGPREP	2	0	1802c	6d69	1			
19060	16COMMIT	2	0	19018	08/27/91 11:01:38				
1a018	16ENDTRANS	2	0	19060	580543				

The following excerpt is taken from the logical log at the database server **apex**:

```

addr  lentype  xid    id link
.....
16018      20BEGIN    2 1 0      08/27/91 10:57:07 3483      pault
1602c      32HINSERT   2      0 16018      1000018 102      4
1604c      68PREPARE   2      0 1602c      eh
17018      16HEURTX    2      0 1604c      1
17028      12CLR       2      0 1602c
17034      16ROLLBACK  2      0 17018      08/27/91 11:01:22
17044      40CKPOINT   0      0 15018      1
      begin      xid      id addr      user
      1          2          1 17034      -----
18018      16ENDTRANS  2      0 17034      8806c3
.....

```

First, you would try to match the transactions in the current database server log with the transactions in the **apex** database server log. The BEGPREP and PREPARE log records each contain the GTRID. You can extract the GTRID by using **onlog -l** and looking at the data portion of the BEGPREP and PREPARE log records. The GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple, though less precise, approach is to look at the time of the COMMIT or ROLLBACK records. The times should be close, although there is a slight delay because of the time taken to transmit the commit (or rollback) message from the coordinator to the participant. (This second approach lacks precision because concurrent transactions could commit at the same time although concurrent transactions from one coordinator would probably not commit at the same time.)

To correct this sample situation

1. Find all records that were updated.
2. Identify their type (insert, delete, update) using **onlog** and the table of record types.
3. Use the **onlog -l** output for each record to obtain the local **xid**, the tblspace number, and the rowid.
4. Map the tblspace number to a table name by comparing the tblspace number to the value in the **partnum** column of the **sysstables** system catalog table.
5. Using your knowledge of the application, determine what action is required to correct the situation.

Example of Manual Recovery

In this example, the time stamps on the COMMIT and ROLLBACK records in the different logs are close. No other active transactions introduce the possibility of another concurrent commit or rollback. In this case, an insert (HINSERT) of assigned rowid 102 hex (258 decimal) was committed on the current database server. Therefore, the compensating transaction is as follows:

```
DELETE FROM t WHERE rowid = 258
```

Notices

IBM may not offer the products, services, or features discussed in this document in all 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 on 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information 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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix[®]; C-ISAM[®]; Foundation.2000[™]; IBM Informix[®] 4GL; IBM Informix[®] DataBlade[®] Module; Client SDK[™]; Cloudscape[™]; Cloudsync[™]; IBM Informix[®] Connect; IBM Informix[®] Driver for JDBC; Dynamic Connect[™]; IBM Informix[®] Dynamic Scalable Architecture[™] (DSA); IBM Informix[®] Dynamic Server[™]; IBM Informix[®] Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix[®] Extended Parallel Server[™]; i.Financial Services[™]; J/Foundation[™]; MaxConnect[™]; Object Translator[™]; Red Brick Decision Server[™]; IBM Informix[®] SE; IBM Informix[®] SQL; InformiXML[™]; RedBack[®]; SystemBuilder[™]; U2[™]; UniData[®]; UniVerse[®]; wintegrate[®] are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Index

Numerics

- 32-bit system support 1-7
- 64-bit addressing
 - buffer pool 7-18
 - database server support 1-7
 - description 7-50
 - maximum number of buffers 7-18
 - memory use 7-50
- 9.3 features, overview Intro-6

A

- Adding
 - listen threads 5-36
 - virtual processors 5-17
- ADM (administration) virtual
 - processor 5-17
- Administrative tasks
 - assigning storage 9-16
 - consistency checking 21-4
 - planning 1-6
 - routine tasks 1-27
- ADT (audit) virtual processor 5-38
- ADTERR parameter 2-17
- ADTMODE parameter 2-17, 5-38
- ADTPATH parameter 2-17
- ADTSIZE parameter 2-17
- Advanced User Rights 1-40
- AFF_NPROCS parameter 5-22
- AFF_SPROC parameter 5-22
- Aggregate, parallel processing
 - of 5-10
- AIO virtual processors 5-31
- ALARMPROGRAM
 - parameter 1-34
- Aliases. *See* DBSERVERALIASES
 - parameter.
- ALL keyword, with
 - DATASKIP 10-39
- ALLOCATE COLLECTION
 - statement 11-10
- ALLOCATE DESCRIPTOR
 - statement 11-10
- ALLOCATE ROW statement 11-10
- Allocating disk space
 - cooked file space 10-8
 - extent 9-15
 - initial chunk 10-10
 - metadata 10-24
 - mirrored data 18-5
 - overview 1-10
 - procedure 10-6
 - sbspaces 9-30
 - shared memory 1-8
- UNIX
 - cooked files 10-8
 - raw files 10-9
- Windows
 - logical drive 10-11
 - NTFS files 10-10
 - physical drive 10-12
- ALTER ACCESS METHOD
 - statement 11-7
- ALTER FRAGMENT
 - statement 11-7
- ALTER FUNCTION
 - statement 11-7
- ALTER INDEX statement 11-7
- ALTER OPAQUE statement 11-8
- ALTER PROCEDURE
 - statement 11-8
- ALTER ROUTINE statement 11-8

ALTER TABLE statement
 changing table type 9-40
 connecting to clients 3-54
 logging 11-8

ANSI compliance
 icon Intro-12
 level Intro-16

ANSI-compliant database
 changing logging mode 12-7
 ondblog utility 12-6
 ontape 12-9

ANSI-compliant transaction
 logging. *See* Logging.

Apache server 4-14

Application, client. *See* Client application.

archecker utility
 overview of 1-28
 See also the *Backup and Restore Guide*.

ASF. *See* Connectivity.

Assertion failure
 data corruption 21-9
 description of 21-3
 determining cause of 21-11
 during consistency checking 21-7
 during processing of user request 21-8
 message log format 21-7

Assertion failure file
 af.xxx 21-7
 gcore.xxx 21-8
 list of 21-7
 shmem.xxx 21-7

Asterisk, as wildcard in hostname
 field 3-50

Asynchronous I/O
 description of 5-29
 kernel (KAIO) 5-27

Atomic data type. *See* Built-in data types.

Attaching to shared memory
 database server utilities 7-11
 description 7-10
 virtual processors 7-12

Auditing
 audit mode 5-38
 configuration parameters 2-17
 overview 1-33

Authentication
 default policy 3-13
 description of 3-12

Automatic database server
 shutdown 1-24
 startup 1-23

Automatic recovery, two-phase
 commit 22-9

Availability
 critical data 9-16
 goal in efficient disk layout 9-52
 sample disk layout 9-55

Average size, smart large
 object 10-24

AVG_LO_SIZE tag 9-26, 10-24

B

Backup
 adding log files 14-17
 blobspaces 10-20
 changing database logging 10-14
 checkpoints 15-18
 chunks 10-17
 converting table type 10-14
 dbspaces 10-15
 definition of 1-28
 deleted log files 14-18
 freeing a log file 14-9
 log files 10-13, 14-6
 physical log 10-13
 RAW tables 9-39
 reducing size of 9-19
 sbspaces 9-31, 10-24
 STANDARD tables 9-39
 strategy 1-6
 table types 9-41
 TEXT and BYTE data 9-20
 verification 1-28

Bad-sector mapping, absence of 21-15

bargroup group 1-11

Before-image
 contents 15-21, 15-24
 described 15-3, 15-4
 flushing of 7-41
 in physical log buffer 7-42
 journal. *See* Physical log.

BEGIN WORK statement 11-9

beginlg field 22-22

Big buffers, description of 7-28

Binary large object. *See* BLOB data type.

Binding CPU virtual
 processors 5-12

Bitmap page
 index tblspace 9-46
 tblspace 9-46

BLOB data type
 mentioned 9-21
 See also Smart large object.

Blobpage
 description 9-11
 determining fullness 10-51
 freeing deleted pages 13-12
 oncheck -pB display 10-51
 physical logging 15-6
 relationship to chunk 9-12
 sizing recommendations 10-21
 storage statistics 10-22
 writes, bypassing shared memory 7-47

Blobspace
 activating 13-12
 adding with
 onspaces 10-19
 backing up 10-13, 10-20
 buffers 7-48
 creating with
 onspaces 10-19
 description 9-20
 dropping with
 initial tasks 10-32
 ON-Monitor 10-33
 onspaces 10-33
 free-map page, tracking
 blobpages 7-49
 logging tasks 9-20, 13-12
 names 10-12
 obtaining the number of free
 blobpages 10-42, 10-43
 restrictions, adding logs 14-16
 storage statistics 10-22
 writing data to 7-47
 writing TEXT and BYTE data 7-48

Block device 9-8

Boldface type Intro-10

Boot file. *See* Startup script.
 B-tree index
 cleaner thread 7-27
 See also the *Performance Guide*.
 Buffer
 64-bit maximum number 7-18
 big buffers 7-28
 blobpage buffer 7-48
 concurrent access 7-40
 current lock-access level for 7-25
 data replication 7-21, 19-11
 dirty 7-40
 exclusive mode 7-34
 flushing 7-40
 how a thread
 accesses a buffer page 7-40
 acquires 7-36
 least-recently used 7-35
 lock types 7-34
 logical-log buffer 7-19
 monitoring statistics and use
 of 8-17
 most-recently used 7-35
 not dirty 7-40
 physical-log buffer 7-20
 share lock 7-34
 smart large objects 7-49, 9-26,
 9-27
 status 7-25
 synchronizing flushing 7-41, 7-42
 threads waiting for 7-25
 write types during flushing 7-43
 Buffer pool
 64-bit addressing 7-18
 bypassed by blobpage data 7-47
 contents of 7-17
 description of 7-17
 flushing 7-41
 full checkpoint 15-12
 fuzzy checkpoint 15-12, 15-13
 LRU queues management 7-36
 minimum requirement 7-18
 monitoring activity 8-20
 read-ahead 7-39
 size of buffer 7-18
 smart large objects 7-49
 synchronizing buffer
 flushing 7-42
 Buffer table, described 7-25

Buffered transaction logging
 when flushed 11-12
 See also Logging.
 BUFFERING tag, onspaces 9-26
 BUFFERS parameter
 64-bit addressing 7-18
 description 7-18
 smart large objects and 7-49
 Buffer-size option 3-41
 BUFSIZE. *See* Page.
 Built-in data types, fuzzy
 operations 15-12
 BYTE data type. *See* TEXT and
 BYTE data; Simple large object.
 Byte-range locking 7-22, 9-27

C

Cache
 data distribution 7-30, 8-8
 monitoring shared-memory
 buffer 8-17, 8-19
 Optical Subsystem memory 10-49
 SPL routine cache
 hash size 2-10, 8-8
 pool size 8-8
 SQL statement cache
 configuration parameters 2-11
 enabling 8-11
 specifying size 8-11
 See also Buffer.
 Calculating size
 blobpages 10-21
 metadata 9-31, 10-25
 page size 10-22
 root dbspace 9-49
 smart large objects 9-26
 Cascading deletes 11-5
 CDR_QDATA_SBFLAGS
 parameter 9-22
 CDR_QDATA_SBSPACE
 parameter 9-22
 Central registry, sqlhosts 3-31
 Changing
 chunk status 20-17
 configuration parameters 2-18
 database server type, HDR 20-21
 logging mode, ANSI
 database 12-7
 Character large object. *See* Smart
 large object.
 Character-special devices 1-9
 Checkpoint
 backup considerations 15-18
 chunk writes 7-44
 data replication 20-17
 description of 4-9, 15-11
 events that initiate 15-14
 flushing of regular buffers 7-41
 forced 20-17
 full 15-12
 fuzzy 15-12
 how it works 15-15 to 15-17
 light appends 10-32
 logical-log buffer 7-19
 logical-log file 15-14
 maximum connections 4-12
 monitoring activity 16-8
 physical-log buffer 7-42, 15-11,
 15-14
 role in data replication 19-16
 role in fast recovery 15-20, 15-21
 step in shared-memory
 initialization 4-9, 4-12
 CHKADJUP log record 10-26,
 10-60
 chkenv utility 1-14
 CHRESERV log record 10-26
 Chunk
 activity during mirror
 recovery 17-9
 adding to
 dbspace 10-17
 mirrored dbspace 18-10
 sbspace 9-30, 10-26
 using ISA or onspaces 10-17
 adding with
 ON-Monitor 10-18
 allocating initial 10-10
 backing up 10-17
 changing mirror chunk
 status 18-7
 checking status 10-41, 10-43,
 21-10
 concepts 9-6
 definition of 1-25

- disk layout guidelines 9-53
- dropping from
 - blob space 10-30
 - db space 10-30
 - sb space 10-30
- exceeding size limits with
 - LVM 9-60
- extents 9-15
- I/O errors during
 - processing 21-10
- linking to the pathname 1-11, 10-9, 10-10, 18-5
- maximum number 10-13
- maximum size 10-13
- monitoring 10-41, 10-44, 21-10
- name, when allocated as raw
 - device 9-8
- recovering a down chunk 18-7
- saving status on secondary
 - server 20-17
- specifying metadata area 10-26
- Chunk free list
 - combining and splitting
 - pages 11-7
 - monitoring 16-4
- Chunk table, description of 7-25
- Chunk write
 - checkpoints 7-44
 - monitoring 8-21
- CKPTINTVL parameter
 - description of 2-10
 - initiating checkpoints 15-14
- Classes of virtual processor 5-5
- CLASSPATH environment
 - variable 1-12, 1-15
- CLEANERS parameter
 - description of 2-11
 - purpose of 7-26
- Client application
 - attaching to shared memory 7-11
 - beginning a session 7-28
 - configuring connectivity 1-16, 1-25, 3-13
 - configuring environment 1-13
 - configuring stack size 7-29
 - connecting to a host 3-29
 - connecting to primary or
 - secondary server 19-5
 - connection type field 3-34
 - connections supported 3-8
 - defined 3-3
 - global transaction 22-4
 - host name field 3-37
 - local-loopback connection 3-12
 - multiplexed connections 3-7
 - network security files 3-17
 - ONCONFIG environment
 - variable 1-13
 - options field 3-40
 - reacting to HDR failure 19-19
 - redirecting in data
 - replication 19-19, 19-21
 - remote hosts 3-17
 - shared-memory connection 3-10
 - specifying a dbservername 3-54
 - sqlhosts entries 3-29, 3-32, 3-50
 - using CSM 3-21, 3-24
 - using data replication 19-5
 - wildcard addressing 3-52
 - Window network domains 3-6
- Client/server configuration
 - example
 - local loopback 3-58
 - multiple connection types 3-60
 - multiple database servers 3-62
 - multiple residency 3-62
 - network connection 3-59
 - shared memory 3-57
 - using IPX/SPX 3-60
 - listen and poll threads 5-33
 - local loopback 3-12
 - shared memory 3-10
- CLIENT_LOCALE environment
 - variable 1-13
- CLOB data type. *See* Smart large object.
- CLOSE statement 11-10
- Code, sample, conventions
 - for Intro-12
- Comment icons Intro-11
- COMMIT statement 11-9, 11-11
- Commit, heterogeneous 22-32
- Communication configuration file.
 - See* ONCONFIG configuration file.
- Communication Support Module
 - concsn.cfg entry 3-23, 3-29
 - CSM configuration file 3-21
 - network security 3-13
 - password encryption 3-20
 - sqlhosts option field 3-43
- Communication support services,
 - message confidentiality and
 - integrity 3-12
- Communications, shared memory
 - description of 7-32
 - how client attaches 7-11
 - size of 7-32
- Communication, client to database
 - server. *See* Connectivity.
- Compliance
 - icons Intro-12
 - with industry standards Intro-16
- concsn.cfg file
 - building SMI tables 3-23
 - description of 3-21
 - entry for network data
 - encryption 3-29
 - entry for password
 - encryption 3-23
 - format of entries 3-21
 - location of 3-21
- Concurrency control 7-33
- Confidentiality, of communication
 - messages 3-12
- Configuration
 - database server environment 1-5
 - db spaces 10-14
 - estimating required disk
 - space 9-52
 - files for network 1-17
 - J/Foundation 1-21
 - monitoring 2-18
 - multiple ports 3-16
 - planning for the database
 - server 1-6
 - requirements 1-5
 - sqlhosts information 1-17
 - storage devices 1-27
 - using Server Setup 1-20
 - Windows 1-8
- Configuration file
 - avoid modifying
 - onconfig.std 1-18
 - connectivity 3-13
 - editing 2-18
 - onconfig.std 1-18

- Configuration parameter
 - ADTERR 2-17
 - ADTMODE 2-17
 - ADTPATH 2-17
 - ADTSIZE 2-17
 - ALARMPROGRAM 1-34
 - BUFFERS 7-18
 - CDR_QDATA_SBFLAGS 9-22
 - CDR_QDATA_SBSpace 9-22
 - CKPTINTVL 2-10, 15-14
 - CLEANERS 2-11
 - CONSOLE 1-37
 - DATASKIP 10-35
 - DBSERVERALIASES 3-54
 - DBSERVERNAME 3-34, 3-54
 - DBSPACETEMP 9-53, 10-16
 - DEF_TABLE_LOCKMODE 2-10, 7-22
 - diagnostic information 21-11
 - DRINTERVAL 2-15
 - DRLOSTFOUND 2-15
 - DRTIMEOUT 2-15
 - DS_HASHSIZE 7-30, 8-8
 - DS_POOLSIZE 7-30, 8-8
 - DUMPCNT 21-11
 - DUMPCORE 21-11
 - DUMPDIR 21-7, 21-11
 - DUMPGCORE 21-8, 21-11
 - DUMPSHMEM 21-7, 21-11
 - DYNAMIC_LOGS 14-17, 14-21, 14-24
 - IDX_RA_PAGES 7-39
 - initial chunk of root dbspace 9-18
 - LOGBUFF 7-20
 - LRUS 7-36
 - LRU_MAX_DIRTY 7-38
 - MIRROROFFSET 10-7
 - MSGPATH 1-35
 - NETTYPE 3-55
 - OFF_RECOVERY_THREADS 2-14
 - ON_RECOVERY_THREADS 2-14
 - OPCACHEMAX 10-50
 - PC_HASHSIZE 2-10, 8-8
 - PC_POOLSIZE 2-10, 8-8
 - RA_THRESHOLD 7-40
 - RESIDENT 8-13
 - ROOTOFFSET 10-7
 - SBSPACE_NAME 9-24, 9-32
 - SBSPACE_TEMP 9-32, 10-28
 - smart LO 10-24
 - SERVENUM 7-12, 7-13
 - setting with a text editor 8-9
 - shared memory 8-3
 - SHMADD 2-9, 7-23
 - SHMBASE 2-9, 7-12, 7-13, 8-6
 - SHMTOTAL 2-9, 7-9
 - SHMVIRTsize 2-9, 7-23
 - STACKSIZE 7-29
 - STMT_CACHE 2-11, 8-11
 - STMT_CACHE_HITS 2-11, 8-11
 - STMT_CACHE_NOLIMIT 2-11, 8-11
 - STMT_CACHE_NUMPOOL 2-11, 8-11
 - STMT_CACHE_SIZE 2-11, 8-11
 - See also the Administrator's Reference; Configuration parameter use; individual parameter names.*
- CONNECT statement
 - example 3-55
 - mentioned 3-54
- Connection
 - database versus network 3-7
 - description of 5-33
 - IPX/SPX 3-60
 - local loopback
 - definition of 3-12
 - example 3-58
 - methods 5-31
 - multiple
 - connection types, example 3-60
 - database servers, example 3-62
 - multiplexed 3-7
 - network, example 3-59
 - security restrictions 3-17
 - shared memory, description of 3-10
 - TCP/IP 3-16, 3-17
 - type field 3-34
- Connection-redirection option 3-42
- Connectivity
 - ASF 3-6
 - configuration parameters 3-53
 - configuring with ISA 1-17
 - files, description of 3-13
 - hosts file 3-15
 - services file 3-15
 - sqlhosts file 1-16
 - UNIX 1-16
 - Windows 1-17
- Consistency checking
 - corruption of data 21-9
 - data and overhead 21-4
 - extents 21-6
 - index corruption 21-10
 - indexes 21-6
 - logical logs 21-6
 - monitoring for data
 - inconsistency 21-7
 - overview 1-28, 21-3
 - periodic tasks 21-4
 - sbspaces 9-31
 - system catalog tables 21-5
 - validating metadata 21-6
- Console messages 1-37
- CONSOLE parameter 1-37
- Constraint, deferred checking 11-5
- Contact information Intro-16
- Contention. *See* Disk contention.
- Context switching
 - description of 5-13
 - how functions when OS controls 5-10
 - OS versus multithreaded 5-10
- Contiguous space for physical log 16-4
- Control structures
 - description of 5-13
 - queues 5-16
 - session control block 5-13
 - stacks 5-15
 - thread control block 5-13
- Conversion, during initialization 4-9
- convert utility 1-10
- Cooked file space
 - allocating 10-8
 - compared with raw space 9-9
 - description of 1-10, 9-9
 - for static data 9-9
- Coordinating database server 22-7
- Core dump
 - contents of gcore.xxx 21-8
 - when useful 21-12

See also DUMPCNT; DUMPDIR;
 DUMPGCORE;
 DUMPSHMEM.
 core file 21-7
 Corruption, resolving I/O
 errors 21-10
 CPU virtual processor
 adding and dropping in online
 mode 5-22, 5-26
 AFF_NPROCS parameter 5-22
 AFF_SPROC parameter 5-22
 binding to 5-12
 DataBlade module on 5-20
 description of 5-19
 how many 5-20
 multiprocessor computer 5-21
 on a single-processor
 computer 5-21
 poll threads 5-32, 5-33
 restrictions on 5-25
 single-processor computer 5-21
 threads 5-4
 types of threads run by 5-19
 user-defined routine on 5-20
 CREATE ACCESS METHOD
 statement 11-8
 CREATE AGGREGATE
 statement 11-8
 CREATE CAST statement 11-8
 CREATE DATABASE
 statement 9-16, 11-8
 CREATE DISTINCT TYPE
 statement 11-8
 CREATE EXTERNAL TABLE
 statement 11-8
 CREATE FUNCTION
 statement 5-25, 11-8
 CREATE INDEX statement 11-8
 CREATE OPAQUE CLASS
 statement 11-8
 CREATE OPCLASS statement 11-8
 CREATE PROCEDURE
 statement 11-8
 CREATE ROLE statement 11-8
 CREATE ROUTINE statement 11-7
 CREATE ROW statement 11-7
 CREATE SCHEMA statement 11-7
 CREATE SYNONYM
 statement 11-7

CREATE TABLE statement
 connecting to clients 3-54
 IN dbspace option 9-16
 logging 11-8
 specifying sbspaces in a PUT
 clause 9-23
 CREATE TEMPORARY TABLE
 statement 11-8
 CREATE TRIGGER statement 11-8
 CREATE VIEW statement 11-8
 Creating
 blobspaces 10-19
 dbspaces 10-14
 ONCONFIG file 1-18
 sbspaces 10-23
 smart large objects 9-23, 10-24
 tables with CLOB or BLOB data
 types 9-31
 temporary dbspaces 10-16
 Critical dbspaces
 mirroring 9-53, 9-58
 storage of 9-16
 Critical section of code
 checkpoints 15-16
 description of 15-3
 cron utility 1-38
 CSM configuration file 3-13, 3-21
 curllog field 22-22

D

Data backup. *See* Backup.
 Data block. *See* Page.
 Data consistency
 fast recovery 15-18
 how achieved 15-3
 monitoring for 21-7
 symptoms of corruption 21-9
 verifying consistency 21-4
 Data definition language (DDL)
 statements 11-7, 19-34
 Data files. *See* Logging.
 Data replication
 Enterprise Replication 1-32
 flush interval 2-15
 High-Availability Data
 Replication, about 1-32
 mentioned 11-5
 read-only mode 4-13
 See also High-Availability Data
 Replication (HDR).
 using database server
 groups 3-44
 wait time for response 2-15
 Data restore. *See* Restore.
 Data storage
 concepts 9-5
 control of 9-16, 9-23
 maximum chunk
 number 10-13
 size 10-13
 maximum chunk size 10-6
 See also Disk space.
 Data Type segment. *See* Disk space.
 Data types
 CLOB and BLOB 9-23, 15-13
 fuzzy operations 15-12
 nonfuzzy operations 15-13
 replicated by HDR 19-4
 user-defined 9-23
 Database
 ANSI-compliant 11-13
 asynchronous I/O 5-29
 description of 9-35
 displaying logging status 12-12
 estimating size of 9-52
 fragmentation 9-37
 location of 9-35
 migration. *See* the *IBM Informix
 Migration Guide*.
 monitoring 10-40, 12-11
 purpose of 9-35
 recovery. *See* Recovery.
 size limits 9-36
 sysutils 4-12
 tuning. *See* Performance tuning.
 See also Recovery.
 Database administrator. *See*
 Administrative tasks.
 Database logging
 backups 10-14
 DTP environment 11-13
 Enterprise Replication 11-5
 Database logging status
 ANSI-compliant 11-12
 changing mode 12-7
 buffered logging 11-12

- canceling logging with
 - ondblog 12-6
- changes permitted 12-4
- changing buffering status
 - using ondblog 12-5
 - using ontape 12-8
 - using SET LOG 11-14
- description of 11-10
- ending logging
 - using ondblog 12-6
 - using ontape 12-8
- making ANSI-compliant
 - using ondblog 12-6
 - using ontape 12-8
- modifying
 - using ISA 12-9
 - using ondblog 12-5
 - using ON-Monitor 12-9
 - using ontape 12-7
- overview 1-29
- setting 11-6
- turning on logging with
 - ontape 12-7
- unbuffered logging 11-11
- Database schema. *See the IBM Informix Migration Guide.*
- Database server
 - 32-bit and 64-bit versions 1-7
 - connecting to multiple 3-62
 - connecting with an SQL API 1-25
 - database creation
 - requirements 11-13
 - groups 3-44
 - message log 1-35
 - multithreaded 5-4
 - Password Communication Support Module 3-23
 - remote 3-18
 - scheduling priority 1-21
- Database server groups
 - Enterprise Replication 3-44
 - HDR 3-44
- Database server name. *See* DBSERVERNAME parameter.
- DATABASE statement 3-54
- DataBlade API
 - accessing smart large objects 9-21, 9-31
 - smart large object size 9-26
- DataBlade modules, virtual processors for 5-20
- Data-distribution cache 7-30, 8-8
- Data-recovery mechanisms
 - fast recovery 15-18
 - smart large objects 9-21
- Data-replication buffer 7-21, 19-11
- DATASKIP parameter
 - ALL keyword 10-39
 - description of 10-35
- Data, estimating disk space for 9-51
- dbexport utility. *See the IBM Informix Migration Guide.*
- dbimport utility. *See the IBM Informix Migration Guide.*
- DBLANG environment variable 1-13
- dbload utility 10-61
- DBPATH environment variable
 - dbserver group 3-46
 - use in automatic redirection 19-22
- dbschema utility. *See the IBM Informix Migration Guide.*
- Dbserver group 3-44
- DBSERVERALIASES parameter
 - description of 3-54
 - example 3-54
 - in sqlhosts file 3-34
 - INFORMIXSERVER 1-12
 - multiple connection types 3-60
- DBSERVERNAME parameter
 - associated protocol 5-32
 - database server ID 2-3
 - description of 3-54
 - INFORMIXSERVER 1-12
 - sqlhosts file 3-34
 - syntax rules 3-34
 - virtual processor for poll thread 5-32
- dbservername.cmd file 1-15, 1-19
- DbSPACE
 - adding a chunk 10-17
 - adding a mirrored chunk 18-10
 - adding with ISA 10-14
 - backing up 10-13, 10-15
 - configuration 10-14
 - creating
 - initial dbSPACE 1-25, 9-18
 - permanent 10-14
 - temporary 10-16
 - using onspaces 10-15
 - description of 9-16
 - dropping
 - chunk from 10-29
 - ON-Monitor 10-33
 - onspaces 10-33
 - overview 10-32
 - link between logical and physical units of storage 9-16
 - mirroring if logical-log files included 17-7
 - monitoring simple large objects 10-52
 - names 10-12
 - purpose of 9-16
 - root dbSPACE defined 9-18
 - shared-memory table 7-26
 - smart large objects 9-13
 - temporary 9-19
- DbSPACE table
 - description of 7-26
 - dropping 10-32
 - metadata 7-26
- DBSPACETEMP environment variable 10-16
- DBSPACETEMP parameter 10-16
- DBSPACETEMP environment variable 9-42
- defining temporary tables 9-53
- load balancing 9-53
- DB_LOCALE environment variable 1-13
- DEADLOCK_TIMEOUT parameter 22-31
- DEALLOCATE DESCRIPTOR statement 11-10
- DEALLOCATE ROW statement 11-10
- DEALLOCATECOLLECTION statement 11-10
- Decision-support query
 - DS_MAX_QUERIES parameter 2-12
 - See also* PDQ (Parallel database query).
- DECLARE statement 11-10
- Default configuration file 1-18, 4-6

- DEFAULT keyword, with SET
 - DATASKIP 10-37
- Default locale Intro-4
- Deferred checking of
 - constraints 11-5
- DEF_TABLE_LOCKMODE
 - parameter 2-10, 7-22
- DELETE statement 11-9
- Deleting
 - log files 14-18
 - RAW tables 9-39
 - smart-large-object data 10-32, 10-33
 - STANDARD tables 9-39
- Dependencies, software Intro-4
- DESCRIBE statement 11-10
- Device
 - character-special 9-8
 - NFS 9-7
 - when offsets are needed 10-7
- Diagnostic information
 - collecting 21-11
 - disk space restraints 21-12
 - parameters to set 21-11
- Diagnostic messages. *See* Message log.
- Dictionary cache 7-30
- Directory, NFS 9-7
- Dirty buffer, description of 7-40
- Dirty Read isolation level 19-34
- Disabling I/O error
 - destructive versus
 - nondestructive 21-12
 - monitoring with
 - event alarms 21-14
 - message log 21-13
 - when they occur 21-13
- Disk allocation 1-10
- Disk configuration 1-7
- Disk contention, reducing 9-52
- Disk I/O
 - errors during processing 21-10
 - kernel asynchronous I/O 5-27
 - logical log 5-27
 - operating system I/O 9-9
 - physical log 5-27
 - priorities 5-27
 - queues 5-31
 - raw I/O 9-8
 - reads from mirrored
 - chunks 17-10
 - role of shared memory in
 - reducing 7-5
 - smart large objects 9-21
 - virtual processor classes 5-27
 - writes to mirrored chunks 17-9
- Disk layout
 - logical volume managers 9-60
 - mirroring 9-53
 - optimum performance 9-53
 - sample disk layouts 9-55
 - table isolation 9-54
 - trade-offs 9-55
- Disk management 1-10
- Disk page
 - before-images in physical
 - log 7-42
 - read ahead 7-39
- Disk space
 - allocating
 - cooked file space 10-8
 - on Windows 10-11
 - raw disk space 10-9
 - creating a link to chunk
 - pathname 1-11, 10-9, 10-10
 - description of 10-6
 - estimating size of 9-49
 - initialization 1-22, 4-3, 4-5, 4-8
 - layout guidelines 9-52
 - maximum
 - chunk number 10-13
 - chunk size 1-25, 10-13
 - middle partitions 9-54
 - monitoring with ISA 10-46
 - offsets for chunk pathnames 10-7
 - requirements 9-52
 - storing TEXT and BYTE
 - data 10-51
 - tracking usage by tblspace 9-45
- Distributed databases 1-33
- Distributed processing 22-4
- Distributed queries
 - defined 1-33
 - sqlhosts setup 1-16
 - two-phase commit 1-33
- Distributed transaction
 - determining if inconsistently
 - implemented 23-6
 - mentioned 11-5
 - two-phase commit protocol 22-6
 - unbuffered logging 11-13
- Distribution statistics 7-30
- Documentation notes Intro-14
- Documentation notes, program
 - item Intro-15
- Documentation, types of Intro-13
 - documentation notes Intro-14
 - machine notes Intro-14
 - release notes Intro-14
- Domain, Windows
 - changing the password 1-40
 - controller 3-5
 - description of 3-5
 - running as the specified user 1-40
 - trusted 3-5
 - user accounts 3-5, 3-6
- DRINTERVAL parameter
 - description of 2-15
 - setting for
 - asynchronous updating 19-12
 - synchronous updating 19-12
- DRLOSTFOUND parameter
 - description of 2-15
 - using data replication 19-13
- DROP ACCESS METHOD
 - statement 11-8
- DROP AGGREGATE
 - statement 11-8
- DROP CAST statement 11-8
- DROP DATABASE statement 11-8
- DROP FUNCTION statement 11-8
- DROP INDEX statement 11-8
- DROP OPCLASS statement 11-8
- DROP PROCEDURE
 - statement 11-8
- DROP ROLE statement 11-8
- DROP ROUTINE statement 11-8
- DROP ROW TYPE statement 11-8
- DROP SYNONYM statement 11-8
- DROP TABLE statement 11-7
- DROP TRIGGER statement 11-7
- DROP TYPE statement 11-7
- DROP VIEW statement 11-7
- Dropping
 - extspaces 10-35
 - sbspaces 10-31
 - storage spaces 10-32

tables in dbspaces 10-32
DRTIMEOUT parameter
description of 2-15
detecting data replication failures 19-18
DSS application. *See* Decision-support query.
DS_HASHSIZE parameter 7-30, 8-8
DS_MAX_QUERIES
parameter 2-12
DS_MAX_SCANS parameter 2-12
DS_POOLSIZE parameter 7-30, 8-8
DS_TOTAL_MEMORY
parameter 2-12, 7-23
DTP. *See* Distributed transaction.
DUMPCNT parameter 21-11
DUMPCORE parameter 21-11
DUMPDIR parameter 21-7, 21-11
DUMPGCORE parameter 21-8, 21-11
DUMPSHMEM parameter 21-7, 21-11
Dynamic Host Configuration
Product 3-15
Dynamic lock allocation 7-22
Dynamic log allocation
description of 14-14
event alarms and messages 14-23
location of files 14-16
log file size 14-15
overview 1-30
Dynamic Server service 1-23
DYNAMIC_LOGS parameter
adding log files 14-17, 14-24
editing value 14-21
enabling and disabling 14-14

E

Editing
ONCONFIG file 1-20
sqlhosts information
UNIX 1-16
Windows 1-17
Encryption, of password 3-20
End-of-group option 3-43

Enterprise Replication
database logging 11-5
HDR, using with 19-9
overview 1-32
RAW tables 9-39
sbspaces 9-22
specifying sbspaces 9-22
STANDARD tables 9-39
TEMP tables 9-40
using database server
groups 1-16, 3-44
See also the *IBM Informix Enterprise Replication Guide*.

Environment
configuration file 1-14
control application 1-15
Environment variable
CLASSPATH 1-12, 1-15
client applications 1-13
CLIENT_LOCALE 1-13
DBLANG 1-13
dbservername.cmd file 1-15
DBSPACETEMP 10-16
DB_LOCALE 1-13
environment-configuration
file 1-14
INFORMIXCONCSMCFG 3-21
INFORMIXDIR 1-12, 1-23
INFORMIXSERVER 1-12
INFORMIXSHMBASE 7-11
INFORMIXSQLHOSTS 1-16, 3-29
informix.rc and .informix
files 1-14
JVPHOME 1-12, 1-15
LD_LIBRARY_PATH 1-15
ONCONFIG 1-13, 2-18
PATH 1-13, 1-23
required 1-12
sample setup file 1-15
SERVER_LOCALE 1-13
setting 1-14
TERM 1-13
TERMCAP 1-13
TERMINFO 1-13
typographical
conventions Intro-10
.profile or .login file 1-14
en_us.8859-1 locale Intro-4

Error messages
I/O errors on a chunk 21-10
two-phase commit protocol 22-23
ESQL/C, accessing smart large
objects 9-21, 9-31
/etc/hosts file 3-13, 19-25
/etc/services file 3-13, 19-25
Event alarm
description 1-34
dynamically added
logs 14-23 to 14-25
Event Viewer, Windows NT 1-38
Example
DBSERVERALIASES
parameter 3-54, 3-55
/etc/services file entry 3-14
how page cleaning begins 7-38
IPX/SPX connection 3-60
local-loopback connection 3-58
multiple connection types 3-60
shared-memory connection 3-57
TCP/IP connection 3-60
Exclusive lock (buffer), description
of 7-34
EXECUTE FUNCTION
statement 11-9
EXECUTE IMMEDIATE
statement 11-9
EXECUTE PROCEDURE
statement 11-9
EXECUTE statement 11-9
Extension virtual processor. *See*
User-defined virtual processor.
Extent
allocating 9-15
description 9-14
how database server
allocates 9-15
initial size 9-14, 9-26
key concepts concerning 9-15
monitoring 10-48
monitoring with sysextents 10-49
next-extent size 9-14, 9-26
purpose of 9-14
relationship to chunk 9-15
sbospace 9-13, 9-25
size for sbospaces 9-13, 9-25
structure 9-14, 9-25
validating 21-6

External backup and restore. *See the Backup and Restore Guide and the Administrator's Reference.*

External space. *See* Extspace.

Extspace

creating 10-34

dropping with onspaces 10-35

F

Fast recovery

cleanup phase 14-17

description of 15-18

details of process 15-20, 15-23

effects of buffered logging 15-19

how database server detects need for 15-19

mentioned 4-9, 4-13, 11-4

no logging 15-19

overview 1-30

purpose of 15-18

sbspaces 9-21

smart large objects 9-33

table types 9-41

when occurs 15-18

Fault tolerance

data replication 19-4, 19-7

fast recovery 15-18

Feature icons Intro-11

Features in 9.3 Intro-6

FETCH statement 11-10

File

configuration 1-18

connectivity configuration 3-13

cooked 1-10

core 21-7

hosts 1-17

hosts.equiv 3-17

JVP properties 1-21

network configuration 1-17

network security 1-17, 3-17

NTFS 9-7

oncfg_servername.servernum 4-10

ONCONFIG 1-18, 4-6

passwd 1-17

perfmon.exe 1-39

permissions 10-8

services 1-17

sqlhosts 1-16

File Allocation Table (FAT)

partitions 1-10

File I/O. *See* Disk I/O.

finderr utility Intro-15

FLRU queues

description of 7-35

See also LRU queues.

FLUSH statement 11-9

Flushing

before-images 7-41

buffers 7-40

data-replication buffer, maximum interval 2-15

Forced residency, setting 4-10

Forcing a checkpoint. *See*

Checkpoint.

Foreground write

before-image 7-41

description of 7-43

monitoring 7-43, 8-21

Formula

logical-log size 14-4

physical-log size 15-7

Fragment

monitoring

disk usage 10-48

I/O requests for 10-44

skipping

selected fragments 10-38

unavailable fragments 10-39

using DATASKIP 10-36

See also Fragmentation.

Fragmentation

over multiple disks 9-54

skipping inaccessible fragments 10-35

tables and indexes 9-37

See also Fragment.

Free list. *See* Chunk free list.

FREE statement 11-10

Freeing log files 14-9 to 14-10

FREE_RE log record 10-60

Full checkpoint

description 15-12

emptying the physical log 15-17

events that initiate 15-14

flushing buffer pool 15-13, 15-17

forcing 16-9, 16-10

last available log 13-11

oldest update 15-15

Fuzzy checkpoint

definition of 15-11

description 15-12

during initialization 4-12

emptying the physical log 15-17

fast recovery 15-23 to 15-28

flushing buffer pool 15-13, 15-17

forcing 16-10

how it works 15-15 to 15-17

logical-log buffer 7-19

oldest update 13-11

physical logging 15-8 to 15-11

physical-log buffer 7-42

Fuzzy operations

buffer pool 15-17

description 15-12

G

Gateway, Informix, in

heterogeneous commit 22-32

gcore file 21-7

GET DESCRIPTOR

statement 11-10

GET DIAGNOSTICS

statement 11-10

Global Language Support

(GLS) Intro-4, 1-13

Global pool, description of 7-32

Global transaction identifier 23-7

GRANT FRAGMENT

statement 11-8

GRANT statement 11-8

Group

database server 3-44

parallel processing of 5-10

GTRID 23-8

H

Hardware mirroring 17-7

Hash table 7-25

hdrmkpri script 20-21

hdrmksec script 20-21

Heaps 7-30

Help Intro-13

- Heterogeneous
 - commit 22-32 to 22-39
- Heuristic decision
 - independent action 22-12
 - types of 22-14
- Heuristic end-transaction
 - description of 22-19
 - determining effects on
 - transaction 23-4
 - illustration and log records 22-29
 - messages returned by 22-21
 - results of 22-21
 - when necessary 22-19
- Heuristic rollback
 - description of 22-15
 - illustration and log records 22-27
 - indications that rollback
 - occurred 23-5
 - long transaction 22-22
 - monitoring, onstat -x 22-22
 - results of 22-17
- High-Availability Data Replication (HDR)
 - actions to take if
 - primary fails 19-20
 - secondary fails 19-19
 - administration of 20-14
 - advantages of 19-5
 - asynchronous updating 19-12
 - changing database server
 - mode 20-20
 - changing database server
 - type 20-21
 - chunk status on secondary 20-17
 - client redirection
 - comparison of methods 19-31
 - DBPATH 19-22
 - handling within an
 - application 19-28
 - INFORMIXSERVER 19-27
 - sqlhosts file 19-23
 - configuring connectivity for 20-5, 20-9
 - data requirements for 20-5
 - DataBlade modules, installing
 - with 19-10
 - description of 19-5
 - designing clients to use
 - primary database server 19-32
 - secondary database
 - server 19-33
 - detecting failures of 19-18
 - DRINTERVAL parameter 19-12
 - DRLOSTFOUND
 - parameter 19-13
 - DRTIMEOUT parameter 19-18
 - Enterprise Replication, using
 - with 19-9
 - hardware and operating-system
 - requirements for 20-4
 - hdrmkpri and hdrmksec
 - scripts 20-21
 - how it works 19-9
 - how updates are replicated 19-10
 - initial replication 19-9
 - lost-and-found transactions 19-13
 - manual switchover 19-20
 - monitoring status 20-22
 - performing DDL
 - operations 19-34
 - planning for 20-3
 - possible failures 19-17
 - restarting after failure 19-20, 20-27, 20-30
 - restoring system after media
 - failure 20-24
 - role of
 - checkpoint 19-16
 - log records 19-10
 - primary database server 19-5
 - secondary database server 19-5
 - temporary dbspaces 19-35
 - setting up 20-4
 - starting 20-9
 - synchronization 19-16
 - synchronous updating 19-12
 - types of data replicated 19-4
 - UDRs, installing with 19-10
 - UDTs, installing with 19-10
 - High-Availability Data Replication (HDR). *See* Data replication.
 - High-Performance Loader (HPL) 9-39, 9-40
 - HKEY_LOCAL_MACHINE
 - registry 1-17
 - host name field
 - description of 3-37
 - IPX/SPX 3-38
 - multiple network interface
 - cards 5-37
 - shared memory 3-38, 3-40
 - syntax rules 3-37
 - using IP addresses 3-48
 - wildcard addressing 3-49
 - Hostname, changing 1-40
 - hosts file 1-17, 3-14, 3-15
 - hosts.equiv file 3-17
 - Hot site backup. *See* Data replication.
 - Hot swap 17-7

 - IBM Informix Client Software
 - Developer's Kit 22-4
 - IBM Informix ODBC Driver 22-4
 - IBM Informix Server Administrator (ISA)
 - adding chunks 10-17, 10-26
 - adding mirrored chunks 18-10
 - changing operating modes 4-14
 - configuring Enterprise
 - Replication 1-16
 - configuring sqlhosts 1-17
 - creating
 - dbspaces 1-25, 10-14
 - sbspaces 10-25
 - database server modes 4-15
 - database-logging status 12-9
 - displaying logging status 12-12
 - editing sqlhosts file 1-16
 - editing the configuration 2-18
 - end mirroring 18-12
 - executing utility commands 8-20
 - monitoring disk storage 10-46
 - monitoring latches or spin
 - locks 8-16
 - overview 1-35
 - recovering a down chunk 18-11
 - Server Setup 1-20
 - shared memory 8-10
 - start mirroring 18-8, 18-9
 - starting 1-20
 - user permissions 4-14
 - using onmode 20-20
 - viewing messages 1-35

- virtual processors 6-5
- IBM Informix Storage Manager (ISM) 1-27
- Icons
 - compliance Intro-12
 - feature Intro-11
 - Important Intro-11
 - platform Intro-11
 - product Intro-11
 - Tip Intro-11
 - Warning Intro-11
- Identifier option 3-46
- ids-example.rc script 1-23
- IDX_RA_PAGES parameter 7-39
- ifx_lo_copy function 9-34
- ifx_lo_specset_flags function 9-34, 10-28
- Ill-behaved user-defined routine 5-25
- imc protocol subfield 3-36
- imcadmin command 3-64
- Impersonate, client 3-19
- Important paragraphs, icon for Intro-11
- Inconsistency, detecting 21-4
- Incremental backup
 - description of 1-28
 - sbspaces 14-7
- Index
 - fragmentation 9-37
 - parallel building of 5-10
 - R-tree 15-7
 - tblspace 9-46
 - validating 21-6
- Industry standards, compliance with Intro-16
- INFO statement 11-10
- .informix file 1-14
- informix user
 - changing password 1-40
 - managing log files 14-3
 - setting permissions 1-10
- Informix-Admin group
 - changing operating modes 4-15
 - file ownership 1-11
 - managing log files 14-3
- INFORMIXCONCSMCFG
 - environment variable 3-21

- \$INFORMIXDIR/etc/sqlhosts. *See* sqlhosts file.
- INFORMIXDIR environment variable
 - description 1-12
 - in shutdown script 1-24
 - in startup script 1-23
- INFORMIXDIR/bin
 - directory Intro-5
- INFORMIXSERVER environment variable
 - client applications 3-46
 - description 1-12
 - multiple versions of the database server 1-24
 - use in client redirection 19-27
- INFORMIXSHMBASE
 - environment variable 7-11
- INFORMIXSQLHOSTS
 - environment variable 1-16, 3-29, 3-31
- INFORMIXSTACKSIZE
 - environment variable 7-29
- informix.rc file 1-14
- Initial configuration
 - creating storage spaces 1-25
 - disk layout 9-52
 - guidelines for root dbspace 9-18
- Initial-extent size 9-14
- Initialization
 - checkpoint 4-9, 4-12
 - configuration files 4-6, 4-9
 - control returned to user 4-11
 - conversion of internal files 4-9
 - disk space 1-22, 4-3, 4-5, 4-8
 - fast recovery 4-9
 - forced residency 4-10
 - maximum connections 4-12
 - message log 4-11
 - oncfg_servername.servernum file 4-10
 - shared memory 4-3, 4-5
 - SMI tables 4-11
 - steps in 4-5
 - sysmaster database 4-11
 - sysutils database 4-12
 - temporary tablespaces 4-10
 - virtual processors 4-8
- INSERT statement 11-9

- Inserting data
 - RAW tables 9-39
 - STANDARD tables 9-39
- Installation
 - MaxConnect 3-63
 - on Windows 1-10
 - sqlhosts registry 3-31
- Instance Manager 1-19, 1-21
- instmgr.exe 1-19
- Integrity of communication messages 3-12
- Integrity, data. *See* Consistency checking.
- Interprocess communications
 - in nettype field 3-35
 - shared memory for 7-5
- IP address
 - how to find 3-49
 - use in hostname field 3-48
- ipcshm protocol and communications portion (shared memory) 7-32
- IPC. *See* Interprocess communications.
- IPX/SPX
 - in hostname field 3-60
 - in servicename field 3-40
 - service, definition of 3-40
 - sqlhosts entry 3-60
- ISO 8859-1 code set Intro-4
- Isolating tables 9-54
- ixpasswd.exe 1-40
- ixsu.exe 1-40
- I/O. *See* Disk I/O.

J

- JAR file 1-21
- Java archive file (JAR) 1-21
- Java configuration parameters 1-21
- Java Development Kit (JDK) 1-12
- Java virtual processor 5-26
- JDBC Driver 1-12
- Join, parallel processing of 5-10
- Journal. *See* Physical log.
- JVP properties file 1-21
- JVPHOME environment variable 1-12, 1-15

J/Foundation
 configuring 1-21
 environment variables 1-15
 JDBC installation directory 1-12
 setting CLASSPATH 1-12

K

KAIO thread 5-27, 5-30
 Keep-alive option 3-46
 Kernel asynchronous I/O
 description of 5-29
 nonlogging disk I/O 5-27
 Kernel parameters, modifying 1-8
 Key value, shared memory 7-13
 krakatoa.jar file 1-12

L

Latch
 monitoring statistics 8-15
 mutex 5-19
 wait queue 5-18
 See also *mutex*. 7-33
 lchwaits field 8-16
 LD_LIBRARY_PATH environment
 variable 1-15
 Level-0 backup, checking
 consistency 21-9
 Levels, backup
 description of 1-28
 sbspaces 14-7
 Licensed users, maximum
 allowed 4-6, 4-12
 Light appends
 physical logging 10-32
 RAW tables 9-39
 STANDARD tables 9-39
 Light scans 7-28
 Lightweight I/O 7-18, 9-27
 Lightweight processes 5-4
 Limits
 chunk number 10-13
 chunk size 10-13
 CPU VPs 5-20
 JVPs 5-26
 user-defined VPs 5-24
 Links, creating 1-11, 10-10

LIO virtual processors 5-28
 Listen threads
 description of 5-33
 multiple interface cards 5-37
 Load balancing
 done by virtual processors 5-9
 performance goal 9-52
 using DBSPACETEMP 9-53
 LOAD statement 10-61, 11-9
 Loading data
 express mode 9-39, 9-40
 methods 10-61
 utilities 10-61
 Local loopback
 connection 3-12, 5-31
 example 3-58
 restriction 3-12
 Locale Intro-4
 Lock table
 configuration 7-22
 contents of 7-22
 description 7-21
 LOCK TABLE statement 11-10
 Locking
 Dirty Read isolation level 19-34
 sbspaces 9-27
 smart large objects 9-21
 Locks
 description of 7-33
 dynamic allocation 7-22
 initial number 7-22
 onstat -k 22-22
 types 7-34
 wait queue 5-18
 Log position 22-22
 LOGBUFF parameter
 logical log buffers 7-20
 smart large objects and 7-49
 LOGFILES parameter
 description 14-21
 editing ONCONFIG value 14-21
 setting logical-log size 14-5
 Logging
 activity that is always logged 11-7
 ANSI compliant databases 12-7
 buffering transaction
 logging 11-11
 database server processes
 requiring 11-4

definition of transaction
 logging 11-6
 displaying status with ISA 12-12
 DTP environment 11-13
 effect of buffering on logical log
 fill rate 13-8
 Enterprise Replication 9-22, 11-5
 metadata and user data 9-33
 monitoring activity 16-6
 physical logging
 description of 15-4
 process of 15-9
 sizing guidelines for 15-6
 suppression in temporary
 dbspaces 9-19
 point-in-time restore 9-39
 process for
 blob space data 13-11, 13-13
 db space data 13-17
 RAW tables 9-39
 role in data replication 19-10
 R-tree indexes 15-7
 sbspaces 9-27, 13-13
 smart large objects 7-49, 9-34,
 13-15
 STANDARD tables 9-39
 suppression for implicit
 tables 9-19
 table types 12-10
 TEXT and BYTE data 13-11
 using transaction logging 11-10
 when to use logging tables 13-14
See also Database logging status.
 Logging database
 RAW tables 9-39
 SQL statements
 always logged 11-7, 11-9
 never logged 11-10
 STANDARD tables 9-39
 table types supported 9-38
 Logging tables, summary of 9-38
 LOGGING tag, onspaces 9-28
 Logical consistency, description
 of 15-20, 15-23
 Logical log
 administrative tasks 1-30, 13-12
 backup schedule 1-27, 1-28
 configuration parameters 14-21
 description of 7-19, 11-4, 13-3

- dynamic allocation 1-30
- global transactions 22-5
- log position 22-22
- logging smart large objects 14-5
- monitoring for fullness using onstat 14-11
- onlog utility 14-23
- performance considerations 13-8
- size guidelines 13-7, 14-4
- types of records 7-19
- validating 21-6
- See also* Logical-log buffer; Logical-log file.
- Logical recovery, data replication 19-11
- Logical units of storage list of 9-6
- Logical volume manager (LVM) description of 9-60
- mirroring alternative 17-6
- Logical volume or unit 1-25
- Logical-log backup checkpoints 15-18
- description of 1-28
- Logical-log buffer checkpoints 7-19
- description of 7-19
- flushing description of 7-45
- logical-log buffer 7-45
- nonlogging databases 7-46
- synchronizing 7-41
- unbuffered logging 7-45
- when a checkpoint occurs 7-46
- when no before-image 7-46
- logical-log records 7-45
- monitoring 16-6
- Logical-log file adding a log file using ON-Monitor 14-18
- using onparams 14-17
- adding log files manually 14-16
- allocating disk space for 13-7
- backup adding log files 14-17
- changing physical schema 10-13
- effect on performance 13-8
- free deleted blobpages 13-12
- goals of 14-6
- cannot add to blobspace or sbspace 14-16
- changing the size of 14-20
- consequences of not freeing 13-10
- deleting 14-18
- description of 11-4, 13-3
- dropping a log file using ON-Monitor 14-19
- using onparams 14-19
- dynamic allocation description 14-14
- file size 14-15
- location of files 14-16
- monitoring 14-23 to 14-25
- size 14-15
- estimating number needed 13-7, 14-5
- event alarms 14-23 to 14-25
- freeing files 13-10, 14-9 to 14-10
- full checkpoint to free 16-10
- I/O to 5-27, 5-28
- LIO virtual processor 5-27, 5-28
- location 13-4, 14-16
- log file number 13-5
- log position 22-22
- minimum and maximum sizes 13-7
- mirroring a dbspace that contains a file 17-7
- moving to another dbspace 14-20
- resizing 14-17
- role in fast recovery 15-20, 15-22 to 15-23, 15-26 to 15-28
- status A 14-8, 14-17, 14-19
- B 14-9 to 14-10
- C 14-10
- D 14-8, 14-19
- description of flags 13-6
- F 14-19
- L 14-10
- U 14-9 to 14-10, 14-19
- switching 13-12, 14-7
- switching to activate blobspaces 13-12
- temporary 14-13
- unique ID number 13-5
- using SMI tables 14-13
- See also* Logical log.
- Logical-log record database server processes requiring 11-4
- SQL statements that generate 11-9
- two-phase commit protocol 22-8, 22-23, 22-26
- Logid 13-5
- logposit field 22-22
- LOGSIZE parameter adding log files 14-17
- changing 14-21
- description 14-21
- increasing log size 14-20
- logical-log size 14-4
- Long transaction consequences of 13-10
- description of 13-9
- heuristic rollback 22-22
- preventing 1-30
- two-phase commit 22-11, 22-17, 22-20
- Loosely-coupled mode 22-4, 22-5
- LO_CREATE_LOG flag 9-34
- LO_CREATE_NOLOG flag 9-34
- LO_CREATE_TEMP flag 9-34, 10-28
- LO_LOG flag 13-15
- LO_NOLOG flag 13-15
- LRU queues buffer pool management 7-36
- configuring multiple 7-36
- description of 7-34
- FLRU queues 7-35
- how database server selects 7-36
- MLRU queues 7-35
- pages in least-recent order 7-35
- LRU write description of 7-44
- monitoring 8-21
- performing 7-44
- triggering of 7-44
- LRUS parameter description of 7-34
- LRU_MAX_DIRTY parameter description 7-38
- example of use 7-38
- how to calculate value 7-38

LRU_MIN_DIRTY
 parameter 7-34
 managing buffer pools 7-38
 LRU_MIN_DIRTY parameter
 default value 7-38
 example of use 7-39
 how to calculate value 7-39
 LRU_MAX_DIRTY
 parameter 7-34
 managing buffer pools 7-38
 LTXEHWM parameter
 description 14-21
 preventing long
 transactions 14-26
 role in heuristic rollback 22-15
 LTXHWM parameter
 description 14-21
 preventing long
 transactions 14-26
 role in heuristic rollback 22-15

M

Machine notes Intro-14
 Manual recovery
 deciding if action needed 23-9
 determining if data
 inconsistent 23-6
 example of 23-10
 obtaining information from
 logical-log files 23-7
 procedure to determine if
 necessary 23-3
 use of GTRID 23-7
 Mapping, bad sector 21-15
 MaxConnect
 description of 3-63
 imc protocol subfield 3-36
 imcadmin command 3-64
 installation 3-63
 monitoring 3-64
 onsocimc protocol 3-37
 ontliimc protocol 3-37
 packet aggregation 3-64
 Maximum
 chunk number 10-13
 chunk size 10-13
 user connections 4-6, 4-12

Media failure
 detecting 17-10
 recovering from 17-4
 Memory
 64-bit platforms 7-50
 adding a segment 8-14
 See also Shared memory.
 Memory Grant Manager. *See your*
 "Performance Guide."
 Message file for error
 messages Intro-15
 Message log
 and data corruption 21-9
 description of 1-35
 during initialization 4-11
 dynamically added logs 14-24
 metadata usage 10-59
 physical recovery 15-5
 viewing the log in ISA 1-35
 Metadata
 allocating 10-24
 and chunks 10-26
 calculating area 9-31, 10-25
 creating 9-26
 dbspace table 7-26
 description of 7-49, 9-22
 dropping sbospace chunks 10-31
 logging 15-6
 moving space from reserved
 area 10-59
 sbospace logging 13-13
 sizing 9-26, 10-25
 temporary sbospace 9-32
 validating 21-6
 MGM. *See your Performance Guide.*
 Microsoft Transaction Server
 (MTS/XA) 22-4
 Migration. *See the IBM Informix*
 Migration Guide.
 Mirror chunk
 adding 18-10
 changing status of 18-7
 creating 18-6
 disk reads from 17-10
 disk writes to 17-9
 recovering 17-11, 18-7
 structure 17-11
 Mirror dbspace
 creating 10-11
 root dbspace 9-18
 MIRROR parameter
 changing 18-4
 initial configuration value 18-4
 Mirroring
 activity during processing 17-9
 alternatives 17-6
 benefits of 17-4
 changing chunk status 18-7
 chunk table 7-25
 costs of 17-4
 creating mirror chunks 18-6
 description of 17-3
 detecting media failures 17-10
 during system initialization 18-7
 enabling 18-4
 ending 18-11
 hardware 17-7
 hot swap 17-7
 if the dbspace holds logical-log
 files 17-7
 network restriction 17-3
 overview 1-29
 recommended disk layout 9-53
 recovering a chunk 18-7
 recovery activity 17-9
 split reads 17-10
 starting 18-3, 18-6, 18-8, 18-9,
 18-10
 status flags 17-8
 steps required 18-3
 stopping 18-12
 what happens during
 processing 17-9
 when mirroring begins 17-7
 when mirroring ends 17-11
 MIRROROFFSET parameter
 root dbspace 9-18
 when needed 10-7
 MIRRORPATH parameter 9-18
 Miscellaneous (MSC) virtual
 processor 5-38
 Mixed transaction result 22-18
 mi_lo_copy function 9-34
 mi_lo_specset_flags function 9-34,
 10-28
 MLRU queues
 description of 7-35
 end of cleaning 7-38

flushing of regular buffers 7-41
how buffer is placed 7-36
how to end page-cleaning 7-39
limiting number of pages 7-38
LRU_MIN_DIRTY
 parameter 7-38
See also LRU queues.

Mode
adding log files 14-17
description of 4-12
dropping log files 14-19
graceful shutdown 4-17
immediate shutdown 4-18
moving log files 14-20
offline from any mode 4-19
offline to online 4-17
offline to quiescent
 gracefully 4-16
online to quiescent
 gracefully 4-17
 immediately 4-18
quiescent to online 4-17
reinitializing shared
 memory 4-16
taking offline 4-19

MODE ANSI keywords, and
 database logging status 11-12

Monitoring
extents 10-49
global transactions 22-22
licensed users 4-6, 4-12
locks 22-22
MaxConnect usage 3-64
memory size 8-6
metadata and user-data
 areas 10-59
number of users 15-7
sbspaces 9-31, 10-55
spin locks 8-16
SQL statement cache 8-11
tblspaces 10-49
user activity 22-22
user connections 4-6, 4-12

Monitoring database server
blob space storage 10-22
buffer-pool activity 8-20
buffers 8-17
checkpoints 16-8
chunks 10-41

configuration parameter
 values 2-18
databases 10-40, 12-11
data-replication status 20-22
disk I/O queues 5-31
extents 10-48
fragmentation disk use 10-44,
 10-48
global transactions 1-34
latches 8-14, 8-15
length of disk I/O queues 5-31
log files 14-10
logging status 12-11
logical-log buffers 16-6
physical-log buffer 7-21, 16-6
physical-log file 16-6
profile of activity 8-15
shared memory 8-14
simple large objects in
 dbspaces 10-52
sources of information 1-34
user threads 7-28
using oncheck 1-36
using ON-Monitor 1-36
using onperf 1-36
using onstat 1-37
using SMI tables 1-37
virtual processors 6-9

Monitoring tools
UNIX 1-38
Windows 1-39

MSGPATH parameter 1-35

MTS/XA 22-4

Multiple concurrent threads
(MCT) 5-12

Multiple connection types
example 3-60
in sqlhosts 3-55
See also Connection.

Multiple network interface
cards 5-37

Multiple residency, example
of 3-62

Multiplexed connection 3-7

Multiprocessor computer
MULTIPROCESSOR
 parameter 5-21
 processor affinity 5-12

MULTIPROCESSOR
 parameter 5-21

Multithreaded processes
description of 5-4
OS resources 5-10

Mutex
description of 5-19, 7-33
synchronization 5-19
when used 7-33

N

Named-pipe connection
description of 3-8, 3-11
platforms supported 3-8

Names, storage spaces 10-12

.netrc file
description of 3-18
sqlhosts security options 3-47

nettype field
format of 3-34
summary of values 3-36
syntax of 3-34
use of interface type 3-60

NETTYPE parameter
for multiplexed connection 3-7
mentioned 3-55
multiple network addresses 5-37
poll threads 5-32
purpose of 3-55
role in specifying a protocol 5-32
VP class entry 5-32

NetWare file server 3-38

Network
configuration files 1-17
security files 1-17

Network communication
how implemented 5-33
types of 5-31
using IPX/SPX 3-38, 3-40, 3-60
using TCP/IP 3-37, 3-38

Network data encryption
CSM configuration file 3-29

Network Information Service 3-15

Network interface cards,
using 5-37

Network protocols, specifying 5-32

Network security
 files 3-17
 hosts.equiv 3-17
 .netrc file 3-18

Network virtual processors
 description of 5-31
 how many 5-33
 poll threads 5-32

New Technology File System (NTFS) 1-10

Next-extent size 9-14

NFS-mounted directory 9-7

NIS servers, effect on /etc/hosts and /etc/services 3-15

Nonfuzzy operations
 description 15-13
 flushing buffer pool 15-13
 flushing buffers 15-15, 15-17

Nonlogging database
 RAW tables 9-39
 SQL statements
 always logged 11-7
 never logged 11-10
 table types supported 9-38

Nonlogging tables 9-38

Nonyielding virtual processor 5-25

ntchname.exe 1-40

NTFS files 9-7
 converting to 1-10

Null file, creating 10-11

NUMCPUVPS parameter 5-33

O

ODBC Driver 22-4

Offset
 definition of 9-9
 prevent overwriting partition information 9-10
 purpose of 10-6
 subdividing partitions 9-10
 when needed 10-6

OFF_RECVRVY_THREADS
 parameter 2-14, 19-15

Oldest update, freeing logical-log file 13-10

OLTP applications. *See* PDQ (Parallel database query).

ON-Bar, setting up 1-27

oncfg_servername.servernum
 file 4-10

oncheck utility
 blobpage information 10-51
 -cc option 21-4, 21-5
 -cD option 21-4
 -ce option 21-4, 21-6
 -cl option 21-4, 21-6
 consistency checking 21-4
 -cR option 21-4, 21-6
 -cr option 21-4
 -cs option 21-4, 21-6
 monitoring metadata and user-
 data areas 10-59
 monitoring sbspaces 10-57
 obtaining information
 blobspaces 10-51, 10-56
 chunks 10-44, 10-45
 configuration 2-18
 extents and
 fragmentation 10-48
 logical logs 14-12
 tblspaces 10-48
 overview 1-36
 -pB option 10-22
 -pe option 10-59
 -pr option 2-18, 14-12, 15-7
 -pS option 13-15
 -ps option 10-58
 validating data pages 21-4
 validating extents 21-4, 21-6
 validating indexes 21-4, 21-6
 validating logs and reserved
 pages 21-4, 21-6
 validating metadata 21-4, 21-6
 validating reserved pages 21-4
 validating system catalog
 tables 21-4, 21-5

ONCONFIG configuration file
 connectivity 3-53
 creating 1-18
 description of 1-18
 displaying with onstat -c 2-18
 during initialization 4-6
 editing 1-20
 Java parameters 1-21
 multiple residency 3-62
 parameters 3-54

ONCONFIG environment variable
 description of 1-13
 multiple database servers 1-24
 setting of 2-18

ONCONFIG file parameters. *See* Configuration parameter.

onconfig.demo file 1-18

ondblog utility
 ANSI compliant database 12-6
 changing logging mode
 ISA 12-9
 ondblog 12-5

oninit utility
 -p option 4-10, 9-33, 9-44
 temporary tables 9-44

Online help Intro-13

Online manuals Intro-13

Online transaction processing. *See* PDQ (Parallel database query).

onload utility. *See* the IBM Informix Migration Guide.

onlog utility
 displaying log records 14-23
 reconstructing a global
 transaction 23-7

onmode utility
 -a option 8-14
 adding a segment 8-14
 -c option 20-17
 changing shared-memory
 residency 8-13
 configuring SQL statement
 cache 8-11
 dropping CPU virtual
 processors 6-8
 -e option 8-11
 forcing a full checkpoint 10-32,
 16-9, 20-17
 freeing a logical-log file 14-10
 killing
 participant thread 22-11
 session 22-15
 transaction 22-12, 22-21, 23-4
 -l option 14-7
 -m option 20-17
 -O option 21-13
 setting database server
 type 20-10, 20-20
 switching logical-log files 14-7

- user thread servicing requests
 - from 5-5
- using in ISA 20-20
- W parameters 8-11
- ON-Monitor
 - adding
 - chunk 10-18
 - logical-log file 14-18
 - mirror chunks 18-10
 - changing database server
 - modes 4-16
 - copying the configuration 2-18
 - creating blobspaces 10-20
 - creating dbspaces 10-15
 - dropping a logical-log file 14-19
 - dropping storage spaces 10-33
 - overview 1-36
 - recovering a chunk 18-11
 - setting parameters
 - performance options 8-10
 - shared memory 8-10
 - virtual processors 6-6
 - starting mirroring 18-9
 - taking a chunk down 18-11
- onparams utility
 - adding a logical-log file 14-17
 - changing physical log
 - location 16-5
 - size 16-5
 - dropping a logical-log file 14-19
- onperf utility 1-36
- onpladm utility 10-61
- onsmsync utility. *See the Backup and Restore Guide.*
- onsocimc protocol 3-37
- onspaces utility
 - a option 10-17, 10-26
 - adding a mirror chunk 18-10
 - adding sbpace chunks 10-26
 - c -b option 10-19
 - c -d option 10-15
 - c -S option 9-23, 10-23
 - c -t option 10-16
 - c -x option 10-34
 - ch option 9-25, 9-28, 9-30, 10-27
 - change chunk status 20-17
 - cl option 10-31
 - creating sbspaces 10-23
 - creating temporary sbpace 9-32
 - d option 10-30, 10-33, 10-35
 - definition of 1-25
 - Df tags 9-25, 9-30
 - dropping sbpace chunks 10-30
 - ending mirroring 18-12
 - f option 10-31, 10-36
 - modifying DATASKIP 10-36
 - recovering a down chunk 18-11
 - s option 20-17
 - starting mirroring 18-9
 - t option 9-53
 - taking a chunk down 18-10
 - U option 10-26
- onstat utility
 - c option 2-18
 - CPU virtual processors 5-20
 - d option 10-55
 - d update option 10-42, 10-43
 - displaying messages 1-35
 - g ath option 6-10
 - g cac stmt 8-11
 - g imc 3-64
 - g seg option 8-6
 - g smb c option 10-54, 10-60
 - g sql option 12-11
 - g ssc options 8-11
 - g stm option 12-11
 - k option 22-22
 - l option 14-17
 - m option 1-35
- monitoring
 - blobspace 10-43
 - buffer use 8-17, 8-18, 8-19
 - buffer-pool 8-21, 8-22
 - chunk status 10-41
 - configuration 2-18
 - data replication 20-22
 - database server profile 8-15, 8-16
 - fragment load 10-44
 - latches 8-15
 - logical-log buffer 16-7
 - logical-log files 14-11, 14-17
 - physical log 16-7
 - shared memory 8-14, 8-15
 - SQL statement cache 8-11
 - SQL statements 12-11
 - transactions 12-11
 - virtual processors 6-10
- overview 1-37
- p option 8-16
- profiling user activity 22-22
- s option 8-15
- temporary sbpace flags 10-28
- tracking
 - global transaction 22-22
 - locks 22-22
- u option 15-7, 22-22
- updating blobpage
 - statistics 10-42, 10-43
- x option 12-11, 22-22
- ontape utility
 - backing up logical-log files 13-10
 - modifying database logging
 - status 12-7
 - setting up 1-27
- ontliimc protocol 3-37
- onunload utility. *See the IBM Informix Migration Guide.*
- ON_RECVR_THREADS
 - parameter 2-14
- OPCACHEMAX parameter
 - configuring memory 10-50
 - description of 2-17
- OPEN statement 11-10
- Operating system
 - 32-bit and 64-bit versions 1-7
 - parameters 8-3
 - tools 1-38
- Operating-system files. *See Cooked file space.*
- Operational tables, altering 12-10
- Optical (OPT) virtual
 - processor 5-38
- Optical Subsystem memory cache
 - allocation 10-50
 - kilobytes of TEXT and BYTE data
 - written 10-50, 10-51
 - number of objects written 10-50, 10-51
 - session ID for user 10-51
 - size 10-50
 - specifying size of 2-17
 - user ID of client 10-51
- Options field
 - buffer-size option 3-41
 - connection-redirection 3-42
 - CSM option 3-43

- end-of-group 3-43
- group option 3-44
- identifier option 3-46
- keep-alive option 3-46
- overview of 3-40
- security option 3-47
- syntax rules 3-40
- OUTPUT statement
- logging 11-10

P

- Packet aggregation 3-64
- Page
 - description 9-10
 - determining database server page size 10-22
 - least-recently used 7-35
 - most-recently used 7-35
 - relationship to chunk 9-10
- Page-cleaner table
 - description of 7-26
 - number of entries 7-26
- Page-cleaner threads
 - alerted during foreground write 7-44
 - description of 7-41
 - flushing buffer pool 7-41
 - flushing of regular buffers 7-41
 - monitoring 7-26
 - role in chunk write 7-44
- PAGE_CONFIG reserved
 - page 2-18, 4-7, 4-9
- PAGE_PZERO reserved page 4-8
- Parallel database query. *See* PDQ.
- Parallel processing, virtual processors 5-10
- Participant database server
 - automatic recovery 22-10
 - description of 22-7
- passwd file 1-17
- Password Communication Support
 - Module 3-20, 3-23
- Password encryption
 - CSM configuration file 3-21, 3-23
 - database server initialization 3-23
 - description of 3-20

- Password, changing for user
 - informix 1-40
- PATH environment variable
 - description of 1-13
 - in shutdown script 1-24
 - in startup script 1-23
- PC_HASHSIZE parameter 2-10, 8-8
- PC_POOLSIZE parameter 2-10, 8-8
- PDQ (Parallel database query)
 - DS_TOTAL_MEMORY
 - parameter 2-12, 7-23
 - overview of 1-32
- Performance
 - capturing data 1-38
 - CPU virtual processors 5-20
 - how frequently buffers are flushed 7-34
 - parameters, setting
 - with ON-Monitor 8-10
 - Performance Monitor 1-39
 - read-ahead 7-39
 - resident shared-memory 7-16
 - shared memory 3-10, 7-5
 - VP-controlled context
 - switching 5-10
 - yielding functions 5-14
 - See also the Performance Guide.*
- Performance enhancements Intro-7
- Performance monitoring tool 1-38
- Performance tuning
 - amount of data logged 15-5
 - disk-layout guidelines 9-52
 - foreground writes 7-43
 - logical volume managers 9-60
 - logical-log size 13-8
 - LRU write 7-44
 - moving the physical log 16-4
 - sample disk layout for optimal performance 9-56
 - spreading data across multiple disks 9-60
- Performing DDL operations on
 - primary server 19-34
- Permissions, setting 1-10, 10-8
- PHYSBUFF parameter 7-21
- PHYSDBS parameter
 - changing size and location 16-5
 - where located 15-6

- PHYSFILE parameter 16-5
- Physical consistency
 - description of 15-12, 15-20
 - fast recovery 15-23
- Physical log
 - backing up 10-13
 - becoming full 15-8
 - before-image contents 15-6
 - buffer 15-9
 - changing size and location
 - possible methods 16-3
 - reasons 16-3
 - restrictions 16-4
 - using a text editor 16-5
 - using ON-Monitor 16-6
 - using onparams 16-5
 - contiguous space 16-4
 - ensuring does not become full 15-8
 - flushing of buffer 15-10
 - how emptied 15-11
 - increasing size 15-7
 - I/O, virtual processors 5-28
 - monitoring 16-6
 - overview 1-30
 - physical recovery messages 15-5
 - reinitialize shared memory 16-3
 - role in fast recovery 15-19, 15-20, 15-21, 15-24
 - sizing guidelines 15-8
 - virtual processors 5-27
 - where located 15-6
- Physical logging
 - activity logged 15-5
 - backups 15-4
 - blobspace blobpages 15-6
 - data buffer 15-10
 - description of 15-4
 - fast recovery 15-4
 - fuzzy checkpoints 15-8 to 15-11
 - light appends 10-32
 - logging details 15-9
 - smart large objects 15-6
- Physical recovery messages 15-5
- Physical units of storage, list of 9-5
- Physical-log buffer
 - checkpoints 7-42, 15-11
 - dbspace logging 15-10
 - description of 7-20

events that prompt flushing 7-42
 flushing of 7-42, 15-10
 monitoring 16-6
 number of 7-21
 PHYSBUFF parameter 7-21
 PIO virtual processors
 description of 5-28
 how many 5-29
 Planning for resources 1-6
 Platform icons Intro-11
 Point-in-time restore
 logging 9-39
 Point-in-time restore. *See the Backup and Restore Guide.*
 Poll threads
 DBSERVERNAME
 parameter 5-32
 description of 5-33
 how many 5-33
 message queues 7-32
 multiple for a protocol 5-32
 nettype entry 5-32
 on CPU or network virtual
 processors 5-32
 Post-decision phase 22-8
 Precommit phase 22-8
 PREPARE statement 11-10
 Preparing ONCONFIG file 1-18
 Presumed-abort
 optimization 22-10
 Primary database server 19-5
 Priority
 aging, preventing 5-22
 disk I/O 5-27
 scheduling 1-21
 Processes
 attaching to shared memory 7-10
 comparison to threads 5-3
 DSA versus dual process
 architecture 5-9
 Processor affinity
 description of 5-12
 using 5-22
 Product icons Intro-11
 Profile statistics 8-15
 Program counter and thread
 data 7-29
 Program group
 Documentation notes Intro-15

Release notes Intro-15
 Protocol, specifying 5-32
 PSWDCSM. *See* Password
 encryption.
 PUT statement 11-9

Q

Query plans 7-30
 Queues
 description of 5-16
 disk I/O 5-31
 ready 5-17
 sleep 5-17
 wait 5-18

R

RAID. *See* Redundant array of
 inexpensive disks.
 Raw disk space
 allocating on UNIX 10-9
 allocating on Windows 9-8
 character-special interface 9-8
 definition of 9-8
 RAW table
 altering 12-10
 backup 9-41
 fast recovery 9-41
 mentioned 9-38
 overview 1-29
 properties of 9-39
 RA_PAGES parameter 7-39
 RA_THRESHOLD parameter 7-40
 Read-ahead
 description of 7-39
 IDX_RA_PAGES parameter 7-39
 RA_THRESHOLD
 parameter 7-40
 when used 7-39
 Read-only mode, description
 of 4-13
 Ready queue
 description of 5-17
 moving a thread to 5-17, 5-18
 Reception buffer 19-11
 Recommendations on
 allocation of disk space 9-9

consistency checking 21-4
 mirroring the physical log 15-7
 Recovery
 from media failure 17-4
 mode, description of 4-13
 parallel processing of 5-10
 RAW tables 9-39
 STANDARD tables 9-39
 two-phase commit protocol 22-9
 Redundant array of inexpensive
 disks (RAID) 17-6
 Referential constraints 11-5
 Registry
 changing the hostname 1-40
 defining multiple network
 addresses 5-36
 Regular buffers
 events that prompt flushing 7-41
 monitoring status of 7-18
 Relay module. *See the IBM Informix Migration Guide.*
 Release notes Intro-14
 Release notes, program
 item Intro-15
 Remote client 3-17
 Remote hosts and clients 3-17
 RENAME COLUMN
 statement 11-8
 RENAME DATABASE
 statement 11-8
 RENAME INDEX statement 11-8
 RENAME TABLE statement 11-8
 Replication server. *See* Data
 replication.
 Requirements, configuration 1-5
 Reserved area
 described 9-22
 monitoring size of 10-54, 10-60
 moving space to metadata
 area 10-59
 Reserved pages
 chunk status for secondary 20-17
 validating 21-6
 Residency. *See* Multiple residency.
 RESIDENT parameter
 during initialization 4-10
 setting with onmode 8-13
 Resident shared memory
 description 4-7, 7-16

- internal tables 7-24
- Resource manager 22-4
- Resource planning 1-7
- Restore
 - RAW tables 9-39
 - STANDARD tables 9-39
 - table types 9-41, 9-42
- Revectoring table 10-7
- REVOKE FRAGMENT
 - statement 11-8
- REVOKE statement 11-8
- .rhosts file 3-18, 1-17
- Roll back
 - fast recovery 15-22, 15-28
 - heuristic, monitoring 22-22
 - RAW tables 9-39
 - smart large objects 9-34
 - SQL statements logged 11-9
 - STANDARD tables 9-39
- Roll forward, fast recovery 15-22, 15-26
- ROLLBACK WORK statement 11-8
- Root dbspace
 - calculating size of 9-49
 - default location 9-18
 - description of 9-18
 - location of logical-log files 13-4
 - mirroring 18-7
 - temporary tables 9-18, 9-45
- ROOTNAME parameter 9-18
- ROOTOFFSET parameter
 - mentioned 9-18
 - when needed 10-7
- ROOTPATH parameter 9-18
- R-tree index, logging 15-7
- runas utility 1-40

S

- Sbpage
 - description of 9-12
 - sizing recommendations 9-12
- Sbspace
 - adding chunks 9-30, 10-26
 - allocating space 9-30
 - altering storage
 - characteristics 10-27
 - backing up 9-31, 10-13, 10-24

- buffering mode 9-26
- calculating metadata 9-31
- characteristics 9-33
- checking consistency 9-31
- creating
 - using ISA 10-25
 - using onspaces 10-23
- defining replication server 9-22
- description of 9-21
- disk structures 9-31
- dropping
 - CLOB and BLOB columns 10-32
 - using onspaces 10-33
- Enterprise Replication 9-22
- fast recovery 9-21
- incremental backups 14-7
- JAR files 1-21
- last-access time 9-27
- lightweight I/O 7-18, 9-27
- locking 9-27
- logging 9-27, 13-13
- metadata 7-49, 9-22, 9-26
- mirroring 10-24
- moving space from reserved
 - area 10-59
- names 10-12
- recoverability 9-21
- reserved area 9-22
- restrictions, adding logs 14-16
- sbpages 9-12
- sizing metadata 10-25
- specifying storage
 - characteristics 9-25, 10-24
- statistics for user-defined
 - data 2-6
- storing smart large objects 7-49, 9-21
- temporary
 - adding chunks 9-33, 10-26
 - backup and restore 9-33
 - creating 10-27
 - description of 9-32
 - dropping chunks 9-33, 10-30
 - dropping sbospace 10-31
 - example 10-28
 - LO_CREATE_TEMP flag 10-28
 - user-data area 7-49, 9-22, 10-26
 - using onstat -g smb c 10-54, 10-60

- SBSPACENAME parameter 9-24, 9-32, 10-24
- SBSPACE TEMP parameter 9-32, 10-28
- Scans
 - indexes 7-39
 - parallel processing of 5-10
 - sequential tables 7-39
- Scheduling priority 1-21
- Secondary database server 19-5, 20-17
- Security
 - files for network 1-17
 - options, sqlhosts 3-47
 - risks with shared-memory
 - communications 3-10
- Segment identifier (shared-memory) 7-13
- Segment. *See* Chunk.
- SELECT INTO TEMP
 - statement 11-9
- SELECT statement 11-10
- Semaphore, UNIX parameters 8-6
- Server Setup 1-20
 - customizing configuration 1-20
- SERVERNUM parameter
 - description of 7-12, 7-13
 - how used 7-12
- SERVER_LOCALE environment
 - variable 1-13
- Service name field in sqlhosts
 - description of 3-38
- IPX/SPX 3-40
- shared memory 3-39
- stream pipes 3-39
- syntax rules 3-37
- services file 1-17, 3-15
- Service, in IPX/SPX 3-40
- Session
 - and active tbspace 7-27
 - and dictionary cache 7-30
 - and locks 7-22
 - and shared memory 7-28
 - and UDR cache 7-31
 - control block 5-13
 - description of 5-13
 - primary thread 7-28
 - shared-memory pool 7-23
 - sqlxexec threads 5-5

- threads 5-5
- Session control block 5-13
 - description of 7-28
 - shared memory 7-28
- SET DATASKIP statement 10-35, 10-36
- setenv.cmd file 1-15
- setnet32 tool 1-17
- Setting up
 - environment variables 1-15
 - ON-Bar 1-27
 - ontape 1-27
- Share lock (buffer), description of 7-34
- Shared data 7-5
- Shared memory
 - allocating 1-8, 4-7, 7-23
 - and blobpages 7-47
 - and SERVERNUM
 - parameter 7-12
 - and SHMBASE parameter 7-12
 - and smart large objects 7-49
 - attaching additional
 - segments 7-12, 7-14, 8-10
 - attaching to 7-10
 - buffer allocation 7-18
 - buffer hash table 7-25
 - buffer locks 7-33
 - buffer pool 7-17, 15-10
 - buffer table 7-24
 - changing residency with
 - onmode 8-13
 - checkpoint 15-11
 - chunk table 7-25
 - communications 3-38, 3-40, 7-32
 - configuration 7-9, 7-23, 8-3
 - contents 7-8, 7-32
 - copying to a file 8-14
 - critical sections 15-3
 - data-distribution cache 7-30
 - data-replication buffer 19-11
 - dbspace table 7-26
 - description 7-5
 - dictionary cache 7-30, 8-10
 - effect of operating-system
 - parameters 8-3
 - first segment 7-12
 - global pool 7-32
 - header 7-14, 7-17
 - heaps 7-30
 - how utilities attach 7-11
 - identifier 7-12
 - initializing 4-3, 4-5, 4-8
 - internal tables 7-24
 - interprocess communication 7-5
 - key value 7-12, 7-13
 - logical-log buffer 7-19
 - lower-boundary address
 - problem 7-14
 - mirror chunk table 7-25
 - monitoring with ISA 8-10
 - monitoring with onstat 8-14
 - mutex 7-33
 - nonresident portion 8-8
 - operating-system segments 7-9
 - overview 1-31
 - page-cleaner table 7-26
 - parameters 2-10
 - performance options 7-5, 8-10
 - physical-log buffer 7-20
 - pools 7-23
 - portions 7-7
 - purposes of 7-5
 - reinitializing 8-12
 - resident portion
 - creating 4-7
 - description of 7-16
 - ISA 8-10
 - ON-Monitor 8-10
 - text editor 8-9
 - segment identifier 7-13
 - semaphore guidelines 8-6
 - session control block 7-28
 - SHMADD parameter 7-23
 - SHMTOTAL parameter 2-9, 7-9
 - SHMVIRTSIZE parameter 2-9, 7-23
 - size of virtual portion 7-23
 - size, displayed by onstat 7-9
 - smart large objects 9-30
 - sorting 7-31
 - SQL statement cache 2-11, 7-8, 7-31, 8-11
 - stacks 7-29
 - STACKSIZE parameter 7-29
 - synchronizing buffer
 - flushing 7-41
 - tables 7-24
 - tblspace table 7-27
 - thread control block 7-28
 - thread data 7-28
 - thread isolation and buffer
 - locks 7-33
 - transaction table 7-27
 - user table 7-27
 - user-defined routine cache 7-31
 - virtual portion 7-22, 7-24, 8-10, 8-14
 - ISA 8-10
- Shared-memory connection
 - example 3-57
 - how a client attaches 7-11
 - in servicename field 3-39
 - message buffers 5-35, 7-32
 - virtual processor 5-31
- SHMADD parameter 2-9, 7-23
- SHMBASE parameter
 - attaching first shared-memory
 - segment 7-12
 - description of 2-9, 7-13
 - setting higher 8-6
 - warning 7-13
- shmenv file, assertion failures 21-7
- shmkey
 - attaching additional
 - segments 7-14
 - description of 7-13
- SHMTOTAL parameter 2-9, 7-9
- SHMVIRTSIZE parameter 2-9, 7-23
- Shutdown
 - automatically 1-24
 - graceful 4-17
 - immediate 4-18
 - mode, description of 4-13
 - script 1-24
 - taking offline 4-19
- Simple large object
 - buffers 7-48
 - creating in a blobspace 7-48
 - descriptor 7-47
 - illustration of blobspace
 - storage 7-49
 - writing to disk 7-47
 - See also* TEXT and BYTE data.
- Simple Password Communication
 - Support Module
 - CSM configuration file 3-23

- description of 3-20
- Single processor computer 5-21
- SINGLE_CPU_VP parameter
 - single processor computer 5-21
- Sizing guidelines
 - logical log 13-7, 14-4
 - physical log 15-8
- Skipping fragments
 - all fragments 10-39
 - all unavailable 10-35, 10-39
 - effect on transactions 10-37
 - selected fragments 10-38
 - when to use feature 10-38
- Sleep queues, description of 5-17
- Sleeping threads
 - forever 5-18
 - types of 5-17
- Smart large object
 - accessing in applications 9-31
 - AVG_LO_SIZE 10-24
 - buffering mode 9-26
 - buffering recommendation 9-27
 - buffers 7-49
 - byte-range locking 7-22
 - calculating average LO size 9-26
 - calculating extent size 9-13, 9-25
 - copying temporary LO 9-34
 - creating 9-23, 10-24
 - data retrieval 9-21
 - dbspaces 9-13
 - deleting CLOB or BLOB
 - data 10-32, 10-33
 - deleting, reference count of
 - 0 10-31
 - description of 9-21
 - Enterprise Replication use 9-22
 - estimated average size 10-26
 - extents and chunks 9-13
 - I/O properties 9-21
 - last-access time 9-27
 - lightweight I/O 7-18, 9-27
 - locking 9-21, 9-27
 - logging 7-49, 9-27
 - memory 9-30
 - metadata 9-22
 - physical logging 15-6
 - sbpages 9-12
 - shared-memory buffer 7-49
 - sizing metadata area 10-59
 - specifying data types 9-23
 - statistics for user-defined
 - data 2-6
 - storage characteristics 9-25, 9-31, 10-24, 10-27
 - turning logging on or off 13-15
 - user data 9-22
 - using logging 13-13
- SMI table
 - aborted table build 4-11
 - and concsm.cfg 3-23
 - during initialization 4-11
 - monitoring
 - buffer pool 8-23
 - buffer use 8-20
 - checkpoints 16-11
 - chunks 10-48
 - data replication 20-24
 - databases 10-40, 12-11
 - extents 10-49
 - latches 8-16
 - log buffer use 16-11
 - logical-log files 14-13
 - shared memory 8-16
 - tblspaces 10-49
 - virtual processors 6-12
 - preparation during
 - initialization 4-11
 - sysextents 10-49
 - systabnames 10-49
 - See also the Administrator's Reference; System-monitoring interface.*
- Sockets in nettype field 3-35
- Software dependencies Intro-4
- Sorting
 - as parallel process 5-10
 - shared memory 7-31
- Spin locks, monitoring 8-16
- SPL routine
 - PC_HASHSIZE parameter 2-10
 - PC_POOLSIZE parameter 2-10
 - UDR cache hash size 2-10
 - UDR cache size 2-10
- SPL routine cache
 - specifying hash size 2-10, 8-8
 - specifying pool size 8-8
 - See also* UDR cache. 7-31
- Split read 17-10
- SPWDCSM 3-20
- SQL statement cache
 - configuring 2-11, 8-11
 - description 7-31
 - monitoring 8-11
 - overview 1-31
 - shared-memory location 7-8
 - specifying size 2-11, 8-11
- SQL statements
 - ALTER TABLE 11-8
 - always logged 11-7
 - CREATE DBSPACE 9-16
 - CREATE TABLE 9-16
 - CREATE TABLE,PUT clause 9-23
 - DECLARE 11-10
 - initial connection to database
 - server 1-25
 - logging databases 11-9
 - monitoring 12-11
 - new features Intro-8
 - not logged 11-10
 - parsing and optimizing 7-31
 - UPDATE STATISTICS 7-30
 - using temporary disk space 9-44
- sqlxec thread
 - and client application 5-13
 - as user thread 5-5
 - role in client/server
 - connection 5-34
- sqlhosts file
 - configuring with ISA 1-17
 - connection type field 3-34
 - CSM option 3-43
 - dbservername field 3-34
 - defining multiple network
 - addresses 5-36
 - description of 1-16, 3-29
 - editing with ISA 1-16
 - end of group option 3-43
 - entries for multiple interface
 - cards 5-37
 - group option 3-44
 - host name field 3-37
 - identifier option 3-46
 - keep-alive option 3-46
 - local-loopback example 3-58
 - multiple connection types
 - example 3-55
 - multiplexed connection 3-7

- network connection example 3-59
 - options field 3-40
 - security options 3-47
 - service name field 3-38
 - shared-memory example 3-57
 - specifying network poll threads 5-32
 - sqlhosts registry
 - description of 3-31
 - INFORMIXSQLHOSTS environment variable 3-31
 - location of 3-31
 - Stack
 - and thread control block 5-16
 - description of 5-15
 - INFORMIXSTACKSIZE environment variable 7-29
 - pointer 5-16
 - size of 7-29
 - STACKSIZE parameter 7-29
 - thread 7-29
 - STACKSIZE parameter
 - changing the stack size 7-29
 - description 7-29
 - Standard database server 19-5
 - STANDARD table
 - altering 12-10
 - backup 9-41
 - fast recovery 9-41
 - mentioned 9-38
 - overview 1-29
 - properties of 9-39
 - restore 9-42
 - START VIOLATIONS TABLE
 - statement 11-8
 - Starting the database server
 - automatically 1-23
 - initializing disk space 1-22
 - starts dbservername command 1-22
 - Startup script 1-24
 - Statistics. *See* onstat utility.
 - Status, log file
 - added 14-8, 14-17, 14-19
 - backed up 14-9 to 14-10
 - checkpoint 14-10
 - current 14-10
 - deleted 14-8, 14-19
 - free 14-19
 - used 14-9 to 14-10, 14-19
 - STMT_CACHE parameter 2-11, 8-11
 - STMT_CACHE_HITS
 - parameter 2-11, 8-11
 - STMT_CACHE_NOLIMIT
 - parameter 2-11, 8-11
 - STMT_CACHE_NUMPOOL
 - parameter 2-11, 8-11
 - STMT_CACHE_SIZE
 - parameter 2-11, 8-11
 - STOP VIOLATIONS TABLE
 - statement 11-8
 - Storage characteristics
 - hierarchy 9-29
 - onspaces -ch 10-27
 - onspaces -Df 10-24
 - sbspaces 9-25
 - smart large objects 9-31, 10-24
 - storage spaces 10-46
 - Storage devices, setup 1-27
 - Storage manager
 - ISM 1-27
 - role in ON-Bar system 1-27
 - Storage space
 - backup schedule 1-27, 1-28
 - definition of 1-25
 - ending mirroring 18-12
 - starting mirroring 18-8, 18-9, 18-10
 - storing NTFS partitions 1-10
 - Storage statistics, blobspaces 10-22
 - Stored procedure. *See* SPL routine.
 - Stored-procedure cache. *See* UDR cache. 7-31
 - stores_demo database Intro-5
 - Stream-pipe connection
 - advantages and disadvantages 3-11
 - in servicename field 3-39
 - Structured Query Language. *See* SQL statements.
 - superstores_demo database Intro-5
 - Swapping memory 7-16
 - Switching between threads 5-16
 - Switching to next log file 14-7
 - Sync checkpoint. *See* Checkpoint.
 - sysdistrib table 7-30
 - syslogs table 14-13
 - sysmaster database
 - creation 4-11
 - SMI tables 1-37
 - See also* SMI table.
 - See* SMI table; System-monitoring interface.
 - sysprofile table 8-15
 - systables, flag values 9-38
 - System catalog tables
 - dictionary cache 7-30
 - location of 9-35
 - optimal storage of 9-54
 - sysdistrib 7-30
 - validating 21-5
 - System console 1-37
 - System failure, defined 15-18
 - System requirements
 - database Intro-4
 - software Intro-4
 - System timer 5-17
 - System-monitoring interface (SMI)
 - using to monitor database server 1-37
 - See also* SMI table.
 - sysutils database, creation 4-12
-
- T**
- Table
 - altering to standard 12-10
 - creating with CLOB or BLOB data types 9-31
 - description of 9-36
 - disk-layout guidelines 9-54
 - dropping 10-32
 - extent and 9-36
 - fragmentation 9-37
 - isolating high-use 9-54
 - middle partitions of disks 9-54
 - migration. *See* the *IBM Informix Migration Guide*.
 - standard 9-39
 - storage considerations 9-37
 - storage on middle partition of disk 9-58
 - temporary
 - cleanup during restart 9-44
 - estimating disk space for 9-51

- types of 1-29
- Table types
 - backing up before
 - converting 10-14
 - backup and restore 9-41
 - fast recovery of 9-41
 - flag values in systables 9-38
 - properties of 9-38
 - RAW 12-10
 - RAW table 9-39
 - restoring 9-42
 - STANDARD table 9-39
 - summary of 9-38
- tail -f command 1-36
- Tblspace
 - contents of table 7-27
 - description of 9-45
 - monitoring with
 - systabnames 10-49
 - temporary tblspace during
 - initialization 4-10
 - types of pages contained in 9-46
- TCP/IP communication protocol
 - in host name field 3-37
 - in service name field 3-38
 - listen port number 3-53
 - using
 - hosts.equiv 3-17
 - internet IP address 3-48
 - IPX/SPX 3-16
 - multiple ports 3-16
 - TCP listen port number 3-53
 - wildcards 3-50
- TEMP table 1-29
- Temporary dbspace
 - amount required for temporary
 - tables 9-51
 - creating 10-16
 - data replication 19-35, 20-6
 - DBSPACETEMP 9-42
 - description 9-19
- Temporary logical logs 14-13
- Temporary sbspace
 - adding chunks 9-33, 10-26
 - backup and restore 9-33
 - characteristics 9-33
 - creating 10-27
 - description of 9-32
 - dropping chunks 9-33, 10-30
 - dropping sbspace 10-31
 - example 10-28
 - LO_CREATE_TEMP flag 10-28
- Temporary smart large object
 - creating 9-34
 - description of 9-34
- Temporary table
 - backup 9-41
 - during initialization 4-10
 - Enterprise Replication 9-40
 - restore 9-42
 - smart large object,
 - temporary 9-34
- TERM environment variable 1-13
- TERMCAP environment
 - variable 1-13
- Terminal interface 1-13
- TERMINFO environment
 - variable 1-13
- TEXT and BYTE data
 - absence of compression 10-61
 - loading 10-61
 - monitoring in a dbspace 10-52
 - writing to a blobspace 7-47
 - See also* Simple large object.
- TEXT data type. *See* TEXT and BYTE data.
- Text editor
 - creating ONCONFIG file 1-18
 - setting configuration parameters
 - performance 8-9
 - shared memory 8-9
 - virtual processors 6-4
- Thread
 - access to resources 5-10
 - accessing shared buffers 7-34
 - concurrency control 7-33
 - context of 5-13
 - control block 5-13, 7-28
 - description of 5-4
 - for client applications 5-3
 - for primary session 5-13
 - for recovery 5-5
 - heaps 7-30
 - how virtual processors
 - service 5-12
 - internal 5-5, 5-19
 - kernel asynchronous I/O 5-30
 - migrating 5-16
 - mirroring 5-5
 - multiple concurrent 5-12
 - page cleaner 5-5, 7-41
 - relationship to a process 5-4
 - scheduling and
 - synchronizing 5-12
 - session 5-5, 5-19
 - sleeping 5-18
 - stacks 7-29
 - switching between 5-16
 - user 5-5
 - waking up 5-17
 - yielding 5-13
- Thread control block
 - creation of 7-28
 - role in context switching 5-14
- Thread-safe virtual processor 5-7
- Tightly-coupled mode 22-5
- Tip icons Intro-11
- TLI. *See* Transport-layer interface.
- TP/XA 1-33, 22-4
- Transaction
 - global
 - definition of 22-4, 22-7
 - determining if implemented
 - consistently 23-4
 - identification number,
 - GTRID 23-8
 - tracking 22-22
 - loosely coupled 22-5
 - mixed result 22-18
 - monitoring 12-11
 - RAW tables 9-40
 - tightly coupled 22-5
- Transaction branches 22-7
- Transaction logging
 - buffered 11-12
 - definition of 11-6
 - Enterprise Replication 9-22
 - unbuffered 11-11
 - when to use 11-10
 - See also* Logging.
- Transaction manager
 - MTS/XA 22-4
 - TP/XA 1-33, 22-4
- Transaction table
 - description of 7-27
 - tracking with onstat 7-27

Transport-layer interface, in
 nettype field 3-35

TRUNCATE TABLE
 statement 11-8

Trusted clients, database server and
 Windows domains 3-6

Trusted domain 3-5

Two-phase commit protocol
 automatic recovery 22-9
 administrator's role 22-10
 coordinator recovery 22-10
 participant recovery 22-10
 configuration parameters
 for 22-31
 contrast with heterogeneous
 commit 22-32
 coordinator recovery 22-7
 DEADLOCK_TIMEOUT 22-31
 description of 22-5
 errors messages for 22-23
 flushing logical log records 22-26
 global transaction definition 22-4,
 22-7
 global transaction identifier 23-8
 heuristic decisions
 heuristic end-transaction 22-19
 heuristic rollback 22-15
 types of 22-14
 independent action 22-14
 definition of 22-11
 errors 22-13
 results of 22-12
 what initiates 22-11
 logical-log records for 22-23
 messages 22-8
 overview 1-33
 participant activity 22-7
 participant recovery 22-10
 post-decision phase 22-8
 precommit phase 22-8
 presumed-abort
 optimization 22-10
 role of current server 22-7
 TXTIMEOUT 22-31
 TXTIMEOUT parameter
 description of 22-31
 onmode -Z 22-21
 two-phase commit protocol 22-31
 Types of buffer writes 7-43

U

UDR cache 7-31

Unbuffered disk access
 compared to buffered 9-8
 in data storage 9-7
 raw disk devices 1-9

Unbuffered logging, flushing the
 logical-log buffer 7-45

Unbuffered transaction logging. *See*
 Logging.

Unique ID 13-5

Units of storage 9-5

UNIX devices
 displaying links to a
 pathname 1-11, 10-10
 ownership, permissions on
 character-special 10-9
 cooked files 10-8

UNIX operating system
 link command 1-11, 10-9
 modifying kernel 1-8
 setting environment
 variables 1-14
 shutdown script 1-24
 startup script 1-23
 terminal interface 1-13

UNLOAD statement 11-9

UNLOCK TABLE statement 11-10

UPDATE statement 11-9

UPDATE STATISTICS
 statement 7-30, 11-8

Updating data
 RAW tables 9-39
 STANDARD tables 9-39

Usability enhancements Intro-6

User accounts and Windows
 domains 3-5

User connections, monitoring 4-6,
 4-12

User data
 creating 10-26
 definition 9-22

User impersonation 3-19

User table
 description of 7-27
 maximum number of entries 7-28

User thread
 acquiring a buffer 7-40

description of 5-5
 in critical sections 15-3
 monitoring 7-28
 tracking 7-27

User-data area, sbospace 7-49

User-defined routine
 ill-behaved 5-25
 Java 1-21
 on nonyielding virtual
 processor 5-25
 virtual processors for 5-20

User-defined routines
 configuring cache 7-31
 memory cache 7-31
 shared-memory location 7-8, 7-32

User-defined routine, parallel
 processing of 5-10

User-defined virtual processor
 how many 5-24
 purpose of 5-24
 running UDRs 7-32
 when to use 5-25

Utility
 attaching to shared-memory 7-11
 chkenv 1-14
 cron 1-38
 executing in ISA 8-20
 iostat 1-38
 managing disk storage 10-5
 onstat
 -d option 10-55
 ps 1-38
 sar 1-38
 UNIX 1-38
 vmstat 1-38
*See also the Administrator's
 Reference; oncheck; ondblog;
 oninit; onlog; onmode;
 onparams; onspaces; onstat;
 ontape; ON-Monitor.*

V

Validating
 data pages 21-5
 extents 21-6
 indexes 21-6
 logical logs 21-6

- metadata 21-6
- reserved pages 21-6
- system catalog tables 21-5
- Virtual portion (shared memory)
 - adding a segment 8-14
 - configuration 7-23, 8-10
 - contents of 7-22, 7-24
 - data-distribution cache 7-30
 - global pool 7-32
 - SHMVRTSIZE parameter 2-9, 7-23
 - SQL statement cache 7-31
 - stacks 7-29
 - UDR cache 7-31
 - virtual extension portion 7-7
- Virtual processor
 - access to shared memory 7-7
 - adding and dropping
 - ISA 6-5
 - ON-Monitor 6-6
 - overview 5-11
 - ADM class 5-17
 - ADT class 5-38
 - advantages 5-9
 - AIO class 5-30, 5-31
 - attaching to shared memory 7-7, 7-12
 - binding to CPUs 5-12
 - classes of 5-5, 5-19
 - context switching 5-10
 - coordination of access to
 - resources 5-10
 - CPU class 5-19
 - description of 5-3
 - disk I/O 5-27
 - dropping (CPU) in online
 - mode 6-8
 - during initialization 4-8
 - extension. *See* User-defined
 - virtual processor.
 - for DataBlade modules 5-20
 - for user-defined routine 5-20
 - how threads serviced 5-12
 - LIO class 5-27, 5-28
 - logical-log I/O 5-28
 - monitoring
 - ISA 6-5
 - onstat utilities 6-9
 - moving a thread 5-9

- MSC (miscellaneous) class 5-38
- multithreaded process 5-4
- network 5-31, 5-33
- nonyielding 5-25
- OPT (optical) class 5-38
- overview 1-31
- parallel processing 5-10
- physical log I/O 5-28
- PIO class 5-27, 5-29
- ready queue 5-17
- setting configuration
 - parameters 6-3
- sharing processing 5-9
- UDR written in Java 5-26
- use of stack 5-15
- user-defined class 5-7, 5-24
- Volume table of contents
 - (VTOC) 10-7
- VP class in NETTYPE
 - parameter 5-32
- VPCLASS parameter
 - configuring CPU VPs 5-20
 - JVPs 5-26
 - mentioned 5-7
 - user-defined VPs 5-25

W

- Wait queue
 - buffer locks 7-34
 - description of 5-18
- Waking up threads 5-17
- Warning
 - files on NIS systems 3-15
 - oncheck -cc output 21-5
- Warning icons Intro-11
- Wildcard addressing
 - by a client application 3-52
 - example 3-50
 - in hostname field 3-49
- Windows
 - allocating raw disk space 10-11
 - automatic startup 1-23
 - configuring memory 1-8
 - convert utility 1-10
 - converting to NTFS 1-10
 - Environment Variables control
 - application 1-15

- Event Viewer 1-38
- Instance Manager 1-21
- ixpasswd utility 1-40
- ixsu utility 1-40
- larger shared memory 8-4
- ntchname utility 1-40
- Performance Monitor 1-39
- setting environment
 - variables 1-15
- setting up connectivity 1-17
- shared memory issues 8-5
- Windows Internet Name
 - Service 3-15
- Write types
 - chunk write 7-44
 - foreground write 7-43
 - LRU write 7-44

X

- X/Open compliance level Intro-16, 1-33, 22-4

Y

- Yielding threads
 - at predetermined point 5-14
 - description of 5-13
 - on some condition 5-14
 - ready queue 5-17
 - switching between 5-12
- ypcat hosts command 3-15
- ypcat services command 3-15

