

Administrator's Guide

for Informix Dynamic Server.2000

Version 9.2
September 1999
Part No. 000-6202

Published by Informix® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

© 1999 Informix Corporation. All rights reserved. The following are trademarks of Informix Corporation or its affiliates, one or more of which may be registered in the United States or other jurisdictions:

Answers OnLine™; C-ISAM®; Client SDK™; DataBlade®; Data Director™; Decision Frontier™; Dynamic Scalable Architecture™; Dynamic Server™; Dynamic Server™, Developer Edition™; Dynamic Server™ with Advanced Decision Support Option™; Dynamic Server™ with Extended Parallel Option™; Dynamic Server™ with MetaCube®; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; Dynamic Virtual Machine™; Enterprise Decision Server™; Formation™; Formation Architect™; Formation Flow Engine™; Gold Mine Data Access®; IIF.2000™; i.Reach™; i.Sell™; Illustra®; Informix®; Informix® 4GL; Informix® InquireSM; Informix® Internet Foundation.2000™; InformixLink®; Informix® Red Brick® Decision Server™; Informix Session Proxy™; Informix® Vista™; InfoShelf™; Interforum™; I-Spy™; Mediazation™; MetaCube®; NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine/Secure Dynamic Server™; OpenCase®; Orca™; PaVER™; Red Brick® and Design; Red Brick® Data Mine™; Red Brick® Mine Builder™; Red Brick® Decisionscape™; Red Brick® Ready™; Red Brick Systems®; Regency Support®, Rely on Red BrickSM; RISQL®, Solution DesignSM; STARindex™; STARjoin™; SuperView®, TARGETindex™; TARGETjoin™; The Data Warehouse Company®; The one with the smartest data wins.™; The world is being digitized. We're indexing it.SM; Universal Data Warehouse Blueprint™; Universal Database Components™; Universal Web Connect™; ViewPoint®; Visionary™; Web Integration Suite™. The Informix logo is registered with the United States Patent and Trademark Office. The DataBlade logo is registered with the United States Patent and Trademark Office.

Documentation Team: Diana Chase, Kathy Eckardt, Virginia Panlasigui, Judith Sherwood, Karen Smith, Litz Suto

GOVERNMENT LICENSE RIGHTS

Software and documentation acquired by or for the US Government are provided with rights as follows:

- (1) if for civilian agency use, with rights as restricted by vendor's standard license, as prescribed in FAR 12.212;
- (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce this legend.

List of Chapters

Section I The Database Server

Chapter 1 Introducing the Database Server

Chapter 2 Overview of Database Server Administration

Section II Configuration

Chapter 3 Installing and Configuring the Database Server

Chapter 4 Configuration Parameters

Chapter 5 Client/Server Communications

Chapter 6 Multiple Residency

Chapter 7 Using Multiple Residency

Section III Modes and Initialization

Chapter 8 Managing Database Server Operating Modes

Chapter 9 Initializing the Database Server

Section IV Disk, Memory, and Process Management

Chapter 10 Virtual Processors and Threads

Chapter 11 Managing Virtual Processors

Chapter 12 Shared Memory

Chapter 13 Managing Shared Memory

Chapter 14 Data Storage

Chapter 15 Managing Disk Space

Chapter 16 Table Fragmentation and PDQ

Section V Logging and Log Administration

Chapter 17 Logging

Chapter 18 Managing Database-Logging Status

Chapter 19 Logical Log

Chapter 20 Managing Logical-Log Files

Chapter 21 Physical Logging

Chapter 22 Managing the Physical Log

Chapter 23 Checkpoints and Fast Recovery

Section VI Fault Tolerance

Chapter 24 Mirroring

Chapter 25 Using Mirroring

Chapter 26 High Availability on Windows NT

Chapter 27 High-Availability Data Replication

Chapter 28 Using High-Availability Data Replication

Chapter 29 Consistency Checking

Section VII Distributed Data

Chapter 30 Multiphase Commit Protocols

Chapter 31 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 Databases	5
New Features	5
Extensibility Enhancements	6
Performance Improvements	6
Special Features	7
Version 9.2 Features from Dynamic Server 7.30.	7
Documentation Conventions	8
Typographical Conventions	8
Icon Conventions	9
Sample-Code Conventions.	11
Additional Documentation	12
On-Line Manuals	12
Printed Manuals	13
On-Line Help	13
Error Message Documentation	13
Documentation Notes, Release Notes, Machine Notes	14
Related Reading	15
Compliance with Industry Standards	15
Informix Welcomes Your Comments	15

Section I The Database Server

Chapter 1 Introducing the Database Server

In This Chapter	1-3
Database Server Users	1-3
End Users	1-3
Application and DataBlade Developers	1-4
Database Administrators	1-4
Database Server Administrators	1-4
Database Server Operators	1-4
Dynamic Server	1-5
Client/Server Architecture	1-5
Dynamic Scalable Architecture	1-8
High Performance	1-11
Fault Tolerance	1-13
Dynamic System Administration	1-17
Database Server Security	1-17
Year 2000 Compliance	1-17
Auditing.	1-17
Distributed Database Queries	1-18
Extensibility	1-18
New Data Types	1-20
Support for Large Objects	1-22
Support for External Spaces	1-23
User-Defined Routines and DataBlade Modules	1-24
User-Defined Virtual Processors	1-24
User-Defined Access Methods	1-25

Chapter 2

Overview of Database Server Administration

In This Chapter	2-3
Database Server Administrator	2-3
Initial Tasks	2-4
Configuration Tasks	2-4
Managing Disk Space.	2-4
Managing Database-Logging Status.	2-5
Managing the Logical Log	2-5
Managing the Physical Log.	2-5
Using Auditing	2-5
Using Mirroring	2-6
Using Data Replication	2-6
Managing Shared Memory	2-6
Managing Virtual Processors	2-7
Managing Parallel Database Query	2-7
Routine Tasks	2-7
Changing Modes	2-7
Backing Up Data and Logical-Log Files	2-8
Monitoring Database Server Activity	2-8
Checking for Consistency	2-8
Summary of Administration Tasks	2-9
Monitoring Database Server Activity	2-12
Sources of Information for Monitoring	2-13
Message Log.	2-13
Event Alarm	2-14
System Console.	2-15
Windows NT Event Viewer.	2-15
ON-Monitor	2-15
SMI Tables	2-15
onstat Utility.	2-16
oncheck Utility	2-17
onperf Tool	2-17

Section II Configuration

Chapter 3 Installing and Configuring the Database Server

In This Chapter	3-3
Planning for the Database Server	3-4
Considering Your Priorities	3-4
Considering Your Environment	3-5
Configuring the Operating System	3-6
Configuring Windows NT	3-6
Modifying UNIX Kernel Parameters	3-7
Allocating Disk Space	3-7
Creating a Raw Device or Unbuffered File	3-8
Creating Standard Device Names	3-8
Setting Permissions, Ownership, and Group	3-9
Installing the Database Server	3-9
Using Multiple Residency.	3-9
Upgrading the Database Server.	3-10
Creating a Private Installation	3-10
Setting Environment Variables	3-10
Required Environment Variables	3-11
Global Language Support.	3-12
Other Environment Variables	3-12
Environment Variable Files	3-13
Configuring Connectivity	3-13
The sqlhosts File	3-14
Connectivity Information on Windows NT.	3-14
Preparing the ONCONFIG Configuration File	3-14
Creating a Configuration File	3-15
Starting and Administering the Database Server	3-17
Preparing to Connect to Applications.	3-17
Starting the Database Server and Initialize Disk Space	3-17
Performing Administrative Tasks	3-18
Setting Up Your Storage Manager and Storage Devices	3-22
Creating Storage Spaces	3-22
Monitoring Configuration Information	3-23
Using Command-Line Utilities	3-23
Using ON-Monitor	3-24

Chapter 4

Configuration Parameters

In This Chapter	4-3
Disk-Space Parameters	4-3
Root Dbspace	4-4
Mirror of Root Dbspace	4-5
Other Space-Management Parameters	4-5
Database Server Identification Parameters	4-6
Logging Parameters	4-7
Logical Log	4-7
Physical Log	4-8
Storage-Space and Logical-Log Backups	4-8
Message-Log Parameters	4-9
Shared-Memory Parameters	4-10
Shared-Memory Size Allocation	4-10
Shared-Memory Space Allocation	4-11
Shared-Memory Buffer Control	4-12
Decision-Support Parameters	4-13
Database Server Process Parameters	4-14
Processor Type	4-14
Processor Affinity	4-15
Time Intervals	4-16
Restore Parameters	4-16
Data-Replication Parameters	4-17
Event-Alarm Parameters	4-17
Dump Parameters	4-18
Specialized Parameters	4-18
Auditing	4-19
Optical Media	4-19
UNIX	4-19

Chapter 5

Client/Server Communications

In This Chapter	5-3
Client/Server Architecture	5-3
Network Protocol	5-4
Network Programming Interface	5-5
Windows NT Network Domain.	5-5
Database Server Connection	5-6
Multiplexed Connection	5-7
Connections That the Database Server Supports	5-9
Local Connections	5-10
Communication Support Services	5-13
Informix Password Communication Support Module	5-14
Connectivity Files	5-14
Network-Configuration Files.	5-15
Network-Security Files.	5-18
CSM Configuration File	5-21
The sqlhosts File and the SQLHOSTS Registry Key	5-23
The sqlhosts Information	5-27
Connectivity Information	5-28
Group Information	5-41
Alternatives for TCP/IP Connections.	5-42
ONCONFIG Parameters for Connectivity	5-46
DBSERVERNAME Configuration Parameter	5-47
DBSERVERALIASES Configuration Parameter	5-47
NETTYPE Configuration Parameter	5-48
Environment Variables for Network Connections	5-49
Examples of Client/Server Configurations	5-49
Using a Shared-Memory Connection	5-50
Using a Local-Loopback Connection	5-51
Using a Network Connection	5-52
Using Multiple Connection Types	5-53
Accessing Multiple Database Servers	5-55

Chapter 6	Multiple Residency	
	In This Chapter	6-3
	Benefits of Multiple Residency	6-3
	How Multiple Residency Works	6-4
	The Role of the ONCONFIG Environment Variable	6-5
	The Role of the SERVERNUM Configuration Parameter	6-5
Chapter 7	Using Multiple Residency	
	In This Chapter	7-3
	Planning for Multiple Residency	7-3
	Creating a New Database Server on Windows NT	7-5
	Adding an Instance of the Database Server	7-6
	Deleting an Instance of the Database Server	7-7
	Creating a New Database Server on UNIX	7-7
	Prepare a Configuration File	7-8
	Set the ONCONFIG Environment Variable	7-8
	Edit the New Configuration File	7-8
	Add Connection Information	7-10
	Update the sqlhosts File	7-10
	Initialize Disk Space	7-10
	Prepare Dbospace and Logical-Log Backup Environment.	7-11
	Update the Operating-System Boot File	7-12
	Check INFORMIXSERVER Environment Variables for Users	7-12

Section III Modes and Initialization

Chapter 8	Managing Database Server Operating Modes	
	In This Chapter	8-3
	Database Server Operating Modes	8-3
	Initializing Disk Space	8-4
	Changing Database Server Operating Modes	8-5
	Users Permitted to Change Modes	8-6
	From Off-Line to Quiescent.	8-6
	From Off-Line to On-Line	8-6
	From Quiescent to On-Line.	8-7
	Gracefully from On-Line to Quiescent	8-7
	Immediately from On-Line to Quiescent	8-8
	From Any Mode Immediately to Off-Line.	8-9

Chapter 9

Initializing the Database Server

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

Section IV Disk, Memory, and Process Management

Chapter 10

Virtual Processors and Threads

In This Chapter	10-3
Virtual Processors	10-3
Threads	10-4
Types of Virtual Processors	10-6
Advantages of Virtual Processors	10-9
How Virtual Processors Service Threads	10-12
Control Structures	10-13
Context Switching	10-14
Stacks.	10-15
Queues	10-16
Mutexes	10-19

Virtual-Processor Classes	10-19
CPU Virtual Processors	10-19
User-Defined Classes of Virtual Processors	10-24
Java Virtual Processors	10-27
Disk I/O Virtual Processors	10-27
Network Virtual Processors	10-32
Communications Support Module Virtual Processor	10-39
Optical Virtual Processor	10-39
Audit Virtual Processor	10-39
Miscellaneous Virtual Processor	10-40

Chapter 11 **Managing Virtual Processors**

In This Chapter	11-3
Setting Virtual-Processor Configuration Parameters	11-3
Setting Virtual-Processor Configuration Parameters with a Text Editor	11-4
Setting Virtual-Processor Configuration Parameters with ON-Monitor.	11-6
Starting and Stopping Virtual Processors	11-8
Adding Virtual Processors in On-Line Mode.	11-8
Dropping CPU Virtual Processors in On-Line Mode	11-11
Dropping User-Defined Virtual Processors in On-Line Mode	11-12
Monitoring Virtual Processors	11-12
Monitoring Virtual Processors with Command-Line Utilities	11-12
Monitoring Virtual Processors with SMI Tables	11-15

Chapter 12 **Shared Memory**

In This Chapter	12-5
Shared Memory	12-5
Shared-Memory Use	12-6
Shared-Memory Allocation.	12-7
Shared-Memory Size	12-9
Action to Take If SHMTOTAL Is Exceeded	12-10
Processes That Attach to Shared Memory	12-11
How a Client Attaches to the Communications Portion	12-11
How Utilities Attach to Shared Memory	12-12
How Virtual Processors Attach to Shared Memory.	12-12
Resident Shared-Memory Segments	12-17

Resident Portion of Shared Memory	12-18
Shared-Memory Header	12-18
Shared-Memory Buffer Pool	12-18
Logical-Log Buffer	12-21
Physical-Log Buffer	12-23
High-Availability Data-Replication Buffer	12-23
Lock Table	12-24
Virtual Portion of Shared Memory	12-25
Management of the Virtual Portion of Shared Memory.	12-25
Components of the Virtual Portion of Shared Memory.	12-26
Communications Portion of Shared Memory	12-33
Virtual-Extension Portion of Shared Memory	12-34
Concurrency Control	12-34
Shared-Memory Mutexes	12-35
Shared-Memory Buffer Locks	12-35
Database Server Thread Access to Shared Buffers	12-36
LRU Queues	12-36
Configuring the Database Server to Read Ahead	12-41
Database Server Thread Access to Buffer Pages	12-42
Flushing Data to Disk	12-45
Events That Prompt Flushing of Buffer-Pool Buffers.	12-46
Flushing Before-Images First.	12-46
Flushing the Physical-Log Buffer	12-46
Synchronizing Buffer Flushing	12-49
Describing Flushing Activity.	12-50
Flushing the Logical-Log Buffer.	12-52
Buffering Large-Object Data	12-55
Writing Simple Large Objects	12-55
Accessing Smart Large Objects	12-58
Memory Use on 64-Bit Platforms	12-59

Chapter 13 **Managing Shared Memory**

In This Chapter	13-3
Setting Operating-System Shared-Memory Configuration Parameters	13-3
Maximum Shared-Memory Segment Size	13-4
Maximum Number of Shared-Memory Identifiers	13-6
Shared-Memory Lower-Boundary Address.	13-6
Maximum Amount of Shared Memory for One Process	13-7
Semaphores	13-7

Setting Database Server Shared-Memory Configuration Parameters	13-8
Setting Parameters for Resident Shared Memory with ON-Monitor.	13-9
Setting Parameters for Resident Shared Memory with a Text Editor	13-11
Setting Parameters for Virtual Shared Memory with ON-Monitor.	13-12
Setting Parameters for Virtual Shared Memory with a Text Editor	13-13
Setting Parameters for Shared-Memory Performance Options with ON-Monitor	13-14
Setting Parameters for Shared-Memory Performance Options with a Text Editor	13-15
Reinitializing Shared Memory	13-16
Turning Residency On or Off for Resident Shared Memory	13-17
Turning Residency On or Off in On-Line Mode	13-17
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory	13-18
Adding a Segment to the Virtual Portion of Shared Memory . . .	13-18
Monitoring Shared Memory	13-19
Monitoring Shared-Memory Segments.	13-19
Monitoring the Shared-Memory Profile	13-20
Monitoring Buffers	13-21
Monitoring Buffer-Pool Activity	13-25
Monitoring Latches	13-28

Chapter 14 **Data Storage**

In This Chapter	14-3
Overview of Data Storage	14-3
Physical Units of Storage	14-5
Chunks	14-5
Disk Allocation for Chunks.	14-6
Pages	14-11
Blobpages.	14-12
Sbpages	14-14
Extents.	14-16

Logical Units of Storage	14-19
Dbspaces	14-19
Blobspaces	14-24
Sbspaces.	14-24
Extspaces	14-35
Databases	14-36
Tables.	14-37
Temporary Tables	14-38
Tblspaces	14-42
Table Fragmentation and Data Storage	14-45
Amount of Disk Space Needed to Store Data	14-45
Size of the Root Dbspace	14-45
Amount of Space That Databases Require	14-48
Disk-Layout Guidelines	14-48
Dbspace and Chunk Guidelines.	14-49
Table-Location Guidelines	14-52
Sample Disk Layouts	14-56
Logical-Volume Manager	14-61

Chapter 15 **Managing Disk Space**

In This Chapter	15-5
Initializing Disk Space	15-6
Allocating Disk Space	15-6
Specifying an Offset.	15-7
Allocating a File for Disk Space on UNIX	15-9
Allocating a File for Disk Space on Windows NT.	15-10
Allocating Raw Disk Space on UNIX	15-10
Allocating Raw Disk Space on Windows NT	15-12
Creating Standard Device Names	15-12
Backing Up After You Change the Physical Schema	15-13
Creating a Dbspace	15-13
Specifying Pathnames for Dbspaces	15-14
Specifying Names and Maximum Number of Storage Spaces	15-15
Backing Up the New Dbspace	15-15
Creating a Temporary Dbspace	15-15
Creating a Dbspace with onspaces.	15-16
Creating a Dbspace with ON-Monitor	15-17

Adding a Chunk to a Dbspace	15-17
Backing Up the New Chunk	15-18
Naming Chunks and Storage Spaces	15-18
Limiting Chunk Size and Number	15-19
Creating a Blobspace	15-20
Backing Up the New Blobspace	15-21
Determining Blobpage Size.	15-21
Creating a Blobspace with onspaces.	15-23
Creating a Blobspace with ON-Monitor	15-23
Adding a Chunk to a Blobspace	15-24
Creating an Sbspace	15-24
Backing Up the New Sbspace	15-25
Sizing Sbspace Metadata	15-25
Adding a Chunk to an Sbspace	15-27
Loading Data Into a Table	15-28
Altering Storage Characteristics of Smart Large Objects	15-28
Dropping a Chunk	15-29
Verifying Whether a Chunk Is Empty	15-29
Dropping a Chunk from a Dbspace with onspaces.	15-30
Dropping a Chunk from a Blobspace	15-30
Dropping a Chunk from an Sbspace with onspaces	15-30
Dropping a Storage Space	15-31
Preparing to Drop a Storage Space	15-31
Backing Up After Dropping a Storage Space	15-32
Dropping a Mirrored Storage Space.	15-32
Dropping a Storage Space with onspaces	15-32
Dropping a Dbspace or Blobspace with ON-Monitor	15-33
Optimizing Blobspace Blobpage Size	15-33
Determining Blobspace Storage Efficiency	15-34
Obtaining Blobspace Storage Statistics	15-34
Determining Blobpage Fullness with oncheck -pB	15-35
Managing Extspaces	15-37
Creating an Extspace	15-37
Dropping a Extspace	15-38

Skipping Inaccessible Fragments	15-38
Using the DATASKIP Configuration Parameter	15-39
Using the Dataskip Feature of onspaces	15-39
Using onstat to Check Dataskip Status	15-39
Using the SQL Statement SET DATASKIP	15-40
Effect of the Dataskip Feature on Transactions.	15-41
Determining When to Use Dataskip	15-41
Monitoring Fragmentation Use	15-42
Displaying Databases	15-43
Using SMI Tables.	15-43
Using ON-Monitor	15-44
Monitoring the Database Server for Disabling I/O Errors	15-44
Using the Message Log to Monitor Disabling I/O Errors	15-44
Using Event Alarms to Monitor Disabling I/O Errors	15-45
Monitoring Disk Usage	15-46
Monitoring Chunks	15-46
Monitoring Tblspaces and Extents	15-53
Monitoring Simple Large Objects in a Blobospace	15-56
Monitoring Simple Large Objects in a Dbspace	15-61
Monitoring Sbspaces	15-62
Features Beyond the Scope of the Database Server	15-63
No Bad-Sector Mapping	15-63
No Compression of TEXT and BYTE Data Types	15-63

Chapter 16 **Table Fragmentation and PDQ**

In This Chapter	16-3
Fragmentation	16-3
Fragmentation Goals	16-5
Responsibility for Fragmentation	16-6
Fragmentation Strategies	16-6
SQL Statements That Perform Fragmentation Tasks.	16-10
Parallel Database Query	16-11
Parallelism	16-12
Structure of a PDQ Query.	16-13
Use of PDQ	16-16
OLTP Applications	16-17
Decision-Support Applications	16-18

Database Server Use of PDQ	16-19
Fragmentation Enhancement to PDQ	16-20
Memory Grant Manager.	16-21
Resource Allocation with PDQ	16-21
Parameters for Controlling PDQ	16-21

Section V Logging and Log Administration

Chapter 17 Logging

In This Chapter	17-3
Database Server Processes That Require Logging	17-3
Transaction Logging	17-6
Database Server Activity That Is Logged	17-6
Activity That Is Always Logged	17-7
Activity Logged for Databases with Transaction Logging	17-8
Logging of Simple Large Objects	17-9
Logging of Smart Large Objects	17-9
Database-Logging Status	17-12
Unbuffered Transaction Logging	17-12
Buffered Transaction Logging	17-13
ANSI-Compliant Transaction Logging	17-13
No Database Logging.	17-14
Databases with Different Log-Buffering Status	17-14
Transaction Logging and Distributed Queries	17-14
Settings or Changes for Logging Status or Mode	17-15

Chapter 18 Managing Database-Logging Status

In This Chapter	18-3
Changing Database-Logging Status	18-4
Modifying Database-Logging Status with ondblog	18-5
Changing Buffering Status with ondblog	18-5
Canceling a Logging Mode Change with ondblog	18-6
Ending Logging with ondblog.	18-6
Making a Database ANSI Compliant with ondblog	18-6

Modifying Database Logging Status with ON-Archive	18-6
Turning On Transaction Logging with ON-Archive	18-7
Canceling a Logging Operation with ON-Archive	18-7
Ending Logging with ON-Archive.	18-8
Changing Buffering Status with ON-Archive	18-8
Making a Database ANSI Compliant with ON-Archive	18-8
Modifying Database Logging Status with ontape	18-9
Turning On Transaction Logging with ontape	18-9
Ending Logging with ontape.	18-9
Changing Buffering Status with ontape	18-10
Making a Database ANSI Compliant with ontape	18-10
Modifying Database Logging Status with ON-Monitor	18-11
Monitoring Transaction Logging	18-11
Monitoring Transaction Logging with SMI Tables	18-11
Monitoring Transaction Logging with ON-Monitor	18-12

Chapter 19

Logical Log

In This Chapter	19-3
Logical Log	19-3
Logical-Log Files	19-4
Logical-Log Administration	19-5
Size of the Logical Log	19-5
Performance Considerations	19-5
Long-Transaction Considerations	19-6
Smart-Large-Object User Data and Long-Transaction Logging	19-6
Size and Number of Logical-Log Files	19-7
Location of Logical-Log Files	19-10
Identification of Logical-Log Files	19-11
Status Flags of Logical-Log Files	19-12
Backup of Logical-Log Files	19-14
Logical-Log Restore	19-14
Point-In-Time Restore	19-14
Freeing of Logical-Log Files	19-15
Database Server Attempt to Free a Log File.	19-15
Action If the Next Logical-Log File Is Not Free	19-16
Logical Log and Long Transactions	19-17
Logs-Full High-Water Mark	19-22
Emergency Log Backup	19-22
Administrative Activity When Logs Need Backing Up.	19-23
Log Space for ON-Archive Tasks	19-24

Logical-Log Administration Tasks for Blobspaces	19-26
Switching Log Files to Activate Blobspaces and Blobspace Chunks	19-26
Backing Up Log Files to Free Blobpages	19-26
Logging Process	19-29
Dbospace Logging	19-29
Blobospace Logging.	19-32
Sbospace Logging	19-32

Chapter 20 **Managing Logical-Log Files**

In This Chapter	20-3
Backing Up Logical-Log Files	20-4
Adding a Logical-Log File	20-4
Adding a Logical-Log File with onparams	20-5
Adding a Logical-Log File with ON-Monitor	20-5
Dropping a Logical-Log File	20-6
Dropping a Logical-Log File with onparams	20-6
Dropping a Logical-Log File with ON-Monitor	20-7
Moving a Logical-Log File to Another Dbospace	20-7
Changing the Size of Logical-Log Files	20-9
Using onparams to Add a Logical-Log File with a New Size	20-9
Using a Text Editor to Change the Size of a Log File	20-9
Changing Logical-Log Configuration Parameters	20-10
Using a Text Editor to Change LOGSIZE or LOGFILES	20-10
Using ON-Monitor to Change LOGSIZE or LOGFILES	20-11
Changing LOGSMAX, LTXHWM, or LTXEHW in the ONCONFIG File	20-12
Changing LOGSMAX, LTXHWM, or LTXEHW with ON-Monitor.	20-12
Freeing a Logical-Log File	20-13
Freeing a Log File with Status A	20-13
Freeing a Log File with Status U	20-13
Freeing a Log File with Status U-B	20-14
Freeing a Log File with Status U-C or U-C-L	20-14
Freeing a Log File with Status U-B-L	20-15
Switching to the Next Logical-Log File	20-15

Monitoring Logging Activity	20-16
Monitoring the Logical Log for Fullness	20-16
Using Command-Line Utilities	20-17
Using ON-Monitor	20-18
Using SMI Tables.	20-19
Monitoring Log-Backup Status	20-20
Displaying Logical-Log Records	20-20

Chapter 21

Physical Logging

In This Chapter	21-3
Physical Logging	21-3
Purpose of Physical Logging.	21-4
Database Server Activity That Is Physically Logged.	21-5
Size and Location of the Physical Log	21-6
Limit to the Size of the Physical Log	21-7
Location of the Physical Log	21-9
Details of Physical Logging	21-9
Page Is Read into the Shared-Memory Buffer Pool	21-10
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	21-10
Change Is Reflected in the Data Buffer	21-10
Physical-Log Buffer Is Flushed to the Physical Log	21-10
Page Buffer Is Flushed	21-11
When a Checkpoint Occurs	21-11
How the Physical Log Is Emptied	21-11

Chapter 22

Managing the Physical Log

In This Chapter	22-3
Changing the Physical-Log Location and Size	22-3
Reasons to Change the Physical-Log Location and Size	22-4
Preparing to Make the Changes.	22-4
Checking For Adequate Contiguous Space	22-4
Using a Text Editor to Change Physical-Log Location or Size	22-5
Using onparams to Change Physical-Log Location or Size	22-5
Using ON-Monitor to Change Physical-Log Location or Size.	22-6
Monitoring Physical and Logical Logging Activity	22-6
Using Command-Line Utilities	22-7
Using SMI Tables.	22-9
Using ON-Monitor	22-9

Chapter 23	Checkpoints and Fast Recovery	
	In This Chapter	23-3
	How the Database Server Achieves Data Consistency	23-3
	Critical Sections	23-4
	Checkpoints	23-4
	Full Checkpoint.	23-5
	Fuzzy Checkpoint	23-5
	Events That Initiate a Fuzzy Checkpoint	23-7
	Events That Initiate a Full Checkpoint	23-7
	Forcing a Full Checkpoint	23-8
	Forcing a Fuzzy Checkpoint	23-9
	Sequence of Events in a Checkpoint.	23-9
	Backup and Restore Considerations.	23-11
	Time Stamps	23-12
	Time Stamps on Disk Pages	23-12
	Time Stamps on Logical-Log Pages	23-12
	Time Stamps on Blobpages	23-13
	Fast Recovery	23-14
	Need for Fast Recovery	23-15
	Situations When Fast Recovery Is Initiated	23-15
	Details of Fast Recovery After A Full Checkpoint	23-16
	Details of Fast Recovery After A Fuzzy Checkpoint	23-20
	Monitoring Checkpoint Information	23-24
	Using onstat Options	23-24
	Using ON-Monitor	23-25
	Using SMI Tables	23-26

Section VI Fault Tolerance

Chapter 24	Mirroring	
	In This Chapter	24-3
	Mirroring	24-4
	Benefits of Mirroring	24-4
	Costs of Mirroring	24-5
	Consequences of Not Mirroring	24-5
	Data to Mirror	24-5
	Alternatives to Mirroring	24-6

Mirroring Process	24-7
Creation of a Mirrored Chunk	24-7
Mirror Status Flags	24-8
Recovery	24-9
Actions During Processing	24-9
Result of Stopping Mirroring.	24-11
Structure of a Mirrored Chunk	24-11

Chapter 25 **Using Mirroring**

In This Chapter	25-3
Steps Required for Mirroring Data	25-3
Enabling Mirroring	25-4
Changing the MIRROR Parameter with ONCONFIG	25-4
Changing the MIRROR Parameter with ON-Monitor	25-5
Allocating Disk Space for Mirrored Data	25-5
Linking Chunks	25-6
Relinking a Chunk to a Device After a Disk Failure	25-6
Using Mirroring	25-7
Mirroring the Root Dbspace During Initialization	25-8
Changing the Mirror Status	25-8
Taking Down a Mirrored Chunk	25-8
Ending Mirroring	25-9
Managing Mirroring	25-9
Starting Mirroring for Unmirrored Dbspaces	25-9
Starting Mirroring for New Dbspaces.	25-10
Adding Mirrored Chunks.	25-11
Taking Down a Mirrored Chunk	25-11
Recovering a Mirrored Chunk	25-12
Ending Mirroring	25-13

Chapter 26 **High Availability on Windows NT**

In This Chapter	26-3
Overview of the MSCS Feature	26-3
How the Database Server Uses MSCS	26-4
Multiple Residency	26-5
Client Connections	26-5
Automatic Backups	26-6
Database Server Administration with MSCS	26-6
The sqlhosts Registry	26-6

Chapter 27

High-Availability Data Replication

In This Chapter	27-3
Data Replication	27-4
High-Availability Data Replication	27-4
How High-Availability Data Replication Works	27-8
How Data Initially Replicates	27-8
Reproduction of Updates to the Primary Database Server	27-9
Threads That Handle High-Availability Data Replication	27-14
Checkpoints Between Database Servers	27-15
How Data Synchronization Is Tracked	27-15
High-Availability Data-Replication Failures	27-16
High-Availability Data-Replication Failures Defined	27-16
Detection of High-Availability Data-Replication Failures	27-17
Actions When a High-Availability Data-Replication Failure Is Detected	27-18
Considerations After High-Availability Data-Replication Failure.	27-18
Redirection and Connectivity for Data-Replication Clients	27-20
Designing Clients for Redirection	27-21
Automatic Redirection with DBPATH	27-21
Administrator-Controlled Redirection with the Connectivity Information	27-23
User-Controlled Redirection with INFORMIXSERVER	27-27
Handling Redirection Within an Application	27-28
Comparison of Different Redirection Mechanisms	27-30
Designing High-Availability Data-Replication Clients	27-31
Setting Lock Mode to Wait for Access to Primary Database Server	27-31
Designing Clients to Use the Secondary Database Server	27-32

Chapter 28

Using High-Availability Data Replication

In This Chapter	28-3
Planning for High-Availability Data Replication	28-4
Configuring a System for High-Availability Data Replication	28-5
Meeting Hardware and Operating-System Requirements	28-5
Meeting Database and Data Requirements	28-6
Meeting Database Server Configuration Requirements	28-6
Configuring High-Availability Data-Replication Connectivity.	28-9
Starting High-Availability Data Replication for the First Time	28-11

Performing Basic Administration Tasks	28-16
Changing Database Server Configuration Parameters	28-16
Backing Up Dbspaces and Logical-Log Files	28-17
Changing the Logging Status of Databases	28-18
Adding and Dropping Chunks, Dbspaces, and Blobspaces	28-18
Using and Changing Mirroring of Chunks	28-18
Managing the Physical Log	28-19
Managing the Logical Log	28-20
Managing Virtual Processors.	28-20
Managing Shared Memory	28-20
Changing the Database Server Mode	28-21
Changing the Database Server Type	28-22
Changing the Type of the Primary Database Server	28-23
Changing the Type of the Secondary Database Server	28-23
Monitoring High-Availability Data-Replication Status	28-24
Restoring Data After Media Failure Occurs	28-26
Restoring After a Media Failure on the Primary Database Server	28-27
Restoring After a Media Failure on the Secondary Database Server	28-28
Restarting High-Availability Data Replication After a Failure	28-29
Restarting After Critical Data Is Damaged	28-30
Restarting If Critical Data Is Not Damaged	28-32

Chapter 29 **Consistency Checking**

In This Chapter	29-3
Performing Periodic Consistency Checking	29-3
Verifying Consistency	29-4
Monitoring for Data Inconsistency.	29-6
Retaining Consistent Level-0 Backups	29-8
Dealing with Corruption	29-8
Symptoms of Corruption	29-9
Fixing Index Corruption	29-9
I/O Errors on a Chunk	29-9
Collecting Diagnostic Information	29-11

Section VII Distributed Data

Chapter 30 Multiphase Commit Protocols

In This Chapter	30-3
Two-Phase Commit Protocol	30-3
When the Two-Phase Commit Protocol Is Used	30-3
Goals for the Two-Phase Commit Protocol	30-4
Two-Phase Commit Concepts	30-5
Phases of the Two-Phase Commit Protocol	30-6
Examples of Two-Phase Commit Transactions	30-8
How the Two-Phase Commit Protocol Handles Failures	30-10
Presumed-Abort Optimization	30-17
Independent Actions	30-18
Situations That Initiate Independent Action	30-19
Possible Results of Independent Action	30-19
The Heuristic Rollback Scenario	30-22
The Heuristic End-Transaction Scenario	30-26
Tracking a Global Transaction	30-28
Two-Phase Commit Protocol Errors	30-29
Two-Phase Commit and Logical-Log Records	30-30
Logical-Log Records When the Transaction Commits	30-31
Logical-Log Records Written During a Heuristic Rollback	30-33
Logical-Log Records Written After a Heuristic End Transaction	30-35
Configuration Parameters Used in Two-Phase Commits	30-37
Function of the DEADLOCK_TIMEOUT Parameter	30-37
Function of the TXTIMEOUT Parameter	30-37
Heterogeneous Commit Protocol	30-38
Gateways That Can Participate in a Heterogeneous Commit Transaction	30-39
Enabling and Disabling Heterogeneous Commit	30-40
How Heterogeneous Commit Works	30-41
Implications of a Failed Heterogeneous Commit	30-43

Chapter 31	Recovering Manually from Failed Two-Phase Commit	
	In This Chapter	31-3
	Determining If Manual Recovery Is Required	31-3
	Determining If a Transaction Was Implemented Inconsistently	31-4
	Determining If the Distributed Database Contains Inconsistent Data	31-6
	Deciding If Action Is Needed to Correct the Situation	31-9
	Example of Manual Recovery	31-10
	Index	

Introduction

In This Introduction	3
About This Manual.	3
Types of Users	3
Software Dependencies	4
Assumptions About Your Locale.	4
Demonstration Databases	5
New Features.	5
Extensibility Enhancements	6
Performance Improvements	6
Special Features	7
Version 9.2 Features from Dynamic Server 7.30.	7
Documentation Conventions	8
Typographical Conventions	8
Icon Conventions	9
Comment Icons	10
Feature, Product, and Platform Icons	10
Sample-Code Conventions.	11
Additional Documentation	12
On-Line Manuals	12
Printed Manuals	13
On-Line Help	13
Error Message Documentation	13
Documentation Notes, Release Notes, Machine Notes	14
Related Reading	15
Compliance with Industry Standards	15
Informix Welcomes Your Comments.	15

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 contains important information on administering Informix Dynamic Server 2000 and provides information about how to install, configure, administer, and use the database server. It describes features, database server concepts, and procedures for performing database server management tasks.

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, see your [Performance Guide](#).

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 assumes 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](#) manual for your database server for a list of supplementary titles.

Software Dependencies

This manual assumes that you are using Informix Dynamic Server 2000, Version 9.2, as your database server.

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a Global Language Support (GLS) locale.

This manual assumes that you use the U.S. 8859-1 English locale as the default locale. The default is **en_us.8859-1** (ISO 8859-1) on UNIX platforms or **en_us.1252** (Microsoft 1252) for Windows NT environments.

This locale supports U.S. English format conventions for dates, times, and currency, and also supports the ISO 8859-1 or Microsoft 1252 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 [Informix Guide to GLS Functionality](#).

Demonstration Databases

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 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 [DB-Access User's Manual](#). For descriptions of the databases and their contents, see the [Informix Guide to SQL: Reference](#).

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

New Features

For a comprehensive list of new database server features, see the release notes. This section lists new features relevant to this manual.

The Version 9.2 features that this manual describes fall into the following areas:

- Extensibility enhancements
- Performance improvements
- Special features
- Version 9.2 features from Version 7.30 of the database server

Extensibility Enhancements

This manual describes the following extensibility enhancements to Version 9.2 of the database server:

- New configuration parameters for the following features:
 - Embedded newline characters in quoted strings
 - User-defined statistics routines
- Enhancements to smart large objects:
 - Round-robin fragmentation for smart large objects
 - ALTER TABLE for smart large objects
- Extensions to utilities:
 - **onstat** for user-defined aggregates
 - **oncheck** and **onlog** for R-tree indexes

Performance Improvements

This manual describes the following performance improvements to Version 9.2 of the database server:

- Enhancements to the database server:
 - Dynamic lock allocation
 - Fuzzy checkpoints
- New configuration parameters for the SQL statement cache
- Extensions to **onstat** for the SQL statement cache

Your [Performance Guide](#) describes additional performance improvements to Version 9.2 of the database server.

Special Features

This manual describes administration considerations for the following special features of Version 9.2 of the database server:

- Long identifiers:
 - 128-character identifier
 - 32-character user names
- Extensions to the ON-Bar utility:
 - ON-Bar object expiration and synchronization tool (**onsmsync**)
 - Verification of ON-Bar backups (**archecker**)
 - External backup and restore

Version 9.2 Features from Dynamic Server 7.30

This manual also describes features first released in Version 7.30. These features fall into the following areas:

- Windows NT-specific features:
 - Multiple residency
 - Raw device support
 - Microsoft cluster support
 - Local-connection support with named pipes
- Manageability: Informix Storage Manager (ISM) to manage the storage devices and media that contain backups
- Connectivity:
 - Greater network security available through support of third-party security services
 - Client interfaces (ESQL/C, Informix ODBC Driver, C++, Java, JDBC, GLS) supported in a single package
- Optical Subsystem shared-library support: no need to order a separate product

- New options for the **oncheck** utility:
 - ❑ The **oncheck -w** option to check and print an index without placing a shared lock on the table
 - ❑ The **oncheck -R** option to check the reserved pages, physical-log pages, and logical-log pages

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.

The following conventions are discussed:

- Typographical conventions
- Icon conventions
- Sample-code conventions

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> <i>italics</i> <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 <i>boldface</i>	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.

(1 of 2)



Convention	Meaning
<code>monospace</code> <i>monospace</i>	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 one or more product- or platform-specific paragraphs.
→	This symbol indicates a menu item. For example, “Choose Tools→Options ” means choose the Options item from the Tools menu.

(2 of 2)

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 the DataBlade API
	Identifies information that is specific to Informix ESQL / C
	Identifies information that relates to the Informix Global Language Support (GLS) feature

(1 of 2)

Icon	Description
 UNIX	Identifies information that is specific to UNIX platforms
 WIN NT	Identifies information that is specific to the Windows NT environment
 WIN NT/95	Identifies information that is specific to Windows NT and Windows 95 environments

(2 of 2)

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 to the indicated feature, product, or platform ends at the next heading at the same or higher level. A ♦ symbol indicates the end of feature-, product-, or platform-specific information that appears within 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 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

For additional information, you might want to refer to the following types of documentation:

- On-line manuals
- Printed manuals
- On-line help
- Error message documentation
- Documentation notes, release notes, and machine notes
- Related reading

On-Line Manuals

An Answers OnLine CD that contains Informix manuals in electronic format is provided with your Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see the installation insert that accompanies Answers OnLine.

Informix on-line manuals are also available on the following Web site:

www.informix.com/answers

Printed Manuals

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com. Please provide the following information when you place your order:

- The documentation that you need
- The quantity that you need
- Your name, address, and telephone number

WIN NT

On-Line Help

Informix provides on-line help with each graphical user interface (GUI) that displays information about those interfaces and the functions that they perform. Use the help facilities that each GUI provides to display the on-line help.

Error Message Documentation

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions.

To read error messages and corrective actions, use one of the following utilities.

Utility	Description
finderr	Displays error messages on line
rofferr	Formats error messages for printing

◆

UNIX

WIN NT

To read error messages and corrective actions in Windows environments, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the Task Bar. ◆

Instructions for using the preceding utilities are available in Answers OnLine. Answers OnLine also provides a listing of error messages and corrective actions in HTML format.

UNIX

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following sections describe the on-line files that supplement the information in this manual. Please examine these files before you begin using your database server. They contain vital information about application and performance issues.

On UNIX platforms, the following on-line files appear in the **\$INFORMIXDIR/release/en_us/0333** directory.

On-Line File	Purpose
ADMINDOC_9.2	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.
SERVERS_9.2	The release notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
IDS_9.2	The machine notes file describes any special actions that you must take to configure and use Informix products on your computer. The machine notes are named for the product described.



WIN NT

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals, along 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 Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

The machine notes do not apply to Windows environments. ♦

Related Reading

For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your [Getting Started](#) manual.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. 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.

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to 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:

`doc@informix.com`

The **doc** alias is reserved exclusively for reporting errors and omissions in our documentation.

We appreciate your suggestions.

The Database Server

- Chapter 1** **Introducing the Database Server**
- Chapter 2** **Overview of Database Server Administration**

Section I



Introducing the Database Server

In This Chapter	1-3
Database Server Users.	1-3
End Users.	1-3
Application and DataBlade Developers	1-4
Database Administrators	1-4
Database Server Administrators	1-4
Database Server Operators	1-4
Dynamic Server	1-5
Client/Server Architecture	1-5
Client Application Types	1-6
Connection to a Database Server	1-7
Dynamic Scalable Architecture	1-8
Parallel-Processing Architecture	1-8
Scalability	1-10
High Performance.	1-11
Dynamic Shared-Memory Management	1-11
Direct Disk Access	1-11
Dynamic Thread Allocation	1-12
Fragmentation and Parallelism	1-13
Fault Tolerance	1-13
Storage Space and Logical-Log Backups of Transaction Records.	1-14
Backup Verification	1-14
External Backup and Restore.	1-15
Fast Recovery	1-15
Point-in-Time Restore	1-15
Restartable Restore	1-16
Mirroring	1-16
High-Availability Data Replication.	1-16

Dynamic System Administration	1-17
Database Server Security	1-17
Year 2000 Compliance	1-17
Auditing	1-17
Distributed Database Queries	1-18
Extensibility	1-18
New Data Types	1-20
User-Defined Data Types	1-21
Complex Data Types	1-21
Data Type and Table Inheritance	1-22
Support for Large Objects	1-22
Simple Large Objects and Blobspaces	1-22
Smart Large Objects and Sbspaces	1-23
Support for External Spaces	1-23
User-Defined Routines and DataBlade Modules	1-24
User-Defined Virtual Processors	1-24
User-Defined Access Methods	1-25

In This Chapter

This chapter describes database server users and introduces Informix Dynamic Server 2000.

Database Server Users

The question “What is the database server?” means different things to different users. The following types of individuals who interact with the database server each have a different perspective:

- End users
- Application developers
- DataBlade developers
- Database administrators
- Database server administrators
- Database server operators

End Users

End users use the *Structured Query Language* (SQL), often embedded in a client application, to access, insert, update, and manage information in databases. These end users might be completely unaware that they are using a database server. To them, the database server is a nameless aspect of the system that they are using.

Application and DataBlade Developers

For the developers of client applications, DataBlade modules, and user-defined routines, the database server offers a number of possibilities for data management, multimedia, isolation levels, and so on.

For features and functions that the database server provides for developers, refer to the [Informix ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#). For information on creating user-defined routines, refer to [Extending Informix Dynamic Server 2000](#). Other volumes, the [Informix Guide to SQL: Reference](#) and the [Informix Guide to SQL: Syntax](#), also provide information that is useful for application developers.

Database Administrators

The *database administrator* (DBA) is primarily responsible for managing access control for a database, as described in “[Database Server Security](#)” on [page 1-17](#). The DBA uses SQL statements to grant and revoke privileges to ensure that the correct individuals are able to perform the actions they need and that untrained or unscrupulous users are kept from performing potentially damaging or inappropriate resource-intensive activities. The [Informix Guide to SQL: Tutorial](#) and the [Informix Guide to SQL: Syntax](#) are also of interest to the DBA.

Database Server Administrators

Unlike the DBA, a database server administrator is responsible for maintenance, administration, and operation of the entire database server, which might manage many individual databases. For a description of the tasks involved in database server administration, refer to [Chapter 2, “Overview of Database Server Administration.”](#)

Database Server Operators

Database server operators are responsible for routine tasks associated with database server administration, including backing up and restoring databases. The same person might fill the roles of administrator and operator.

Dynamic Server

Dynamic Server is a database server in an *object-relational database-management system* (ORDBMS) that combines object-oriented and relational capability yet still represents all data in the form of tables with rows and columns. In addition to providing support for built-in alphanumeric data such as character strings, integers, decimals, and dates, the database server provides object-oriented capabilities that enable you to add new data types and supporting routines to *extend* its data management capabilities.

The database server provides the following features:

- Client/server architecture
- Dynamic Scalable Architecture
- High performance
- Fault tolerance and high availability
- Dynamic system administration
- Database server security
- Year 2000 compliance
- Auditing ♦
- Distributed database queries
- Extensibility

The following sections explain each of these features.

Client/Server Architecture

The *database server* provides reliable access to a database for client application programs. A *client* is an application program that a user runs to request information from a database that the database server manages.

When a client application connects to the database server and requests information, the database server locates the requested data within its databases and sends back the results. Depending on the type of request that the client application issues, the database server can return selected rows from tables within the databases that it manages, add or delete rows, or update particular columns within selected rows.

Client applications use Structured Query Language (SQL) to send requests for data to the database server. Client applications can use the data that the database server returns in a variety of ways, including simple displays on computer screens or complicated reports. Client programs include the DB-Access utility and programs that you write with an Informix application programming interface (API) such as Informix ESQL/C, Informix ODBC Driver, C++, or Java.

The database server performs additional activities such as coordinating concurrent requests from multiple clients and enforcing physical and logical consistency on the data to provide reliable access to the database.

The database server administrator starts the database server processes. After the database server is initialized, database server processes run continuously during the period that users access data. For more information on starting the database server, refer to [Chapter 9, “Initializing the Database Server.”](#)

Client Application Types

Two broad classes of applications operate on data that is stored in an object-relational database:

- On-line transaction processing (OLTP) applications
- Decision-support system (DSS) applications

OLTP Applications

OLTP applications are often used to capture new data or update existing data. These operations usually involve quick, indexed access to a small number of rows. An order-entry system is a typical example of an OLTP application. OLTP applications are usually multiuser applications with acceptable response times measured in fractions of a second.

OLTP applications have the following characteristics:

- Simple transactions that involve small amounts of data
- Indexed access to data
- Many users
- Frequent requests
- Fast response times

DSS Applications

DSS applications are often used to report on or consolidate data that has been captured through OLTP operations over time. These applications provide information that is often used for accounting, strategic planning, and decision-making. Data within the database is usually queried but not updated during DSS operations. Typical DSS applications include payroll, inventory, and financial reports.

A recent approach to DSS consolidates enterprisewide data in a separately designed environment, commonly called a *data warehouse*. A data warehouse stores business data for a company in a single, integrated relational database that provides a historical perspective on information for DSS applications.

Another approach to DSS operations, called a *data mart*, draws selected data from OLTP operations or a data warehouse to answer specific types of questions or to support a specific department or initiative within an organization.

Decision-support applications have the following characteristics:

- Complex queries that involve large amounts of data
- Large memory requirements
- Few users
- Periodic requests
- Relatively long response times

Connection to a Database Server

A client application communicates with the database server through the connection features that the database server provides.

At the source-code level, a client application connects to the database server through an SQL statement. Beyond that, the client use of connection facilities is transparent to the application. Library functions that are automatically included when a client program is compiled enable the client to connect to the database server.

As the database administrator, you specify the database server names and types of connections that can be made. On UNIX, the connectivity information is in the **sqlhosts** file. On Windows NT, the connectivity information is in the SQLHOSTS registry key.

To connect to the database server, the client specifies a database server name, which is a logical name assigned by the database administrator. The DBA associates this name with physical characteristics stored in **sqlhosts**, such as the host name, connection type and port number.

For a description of these connection features, refer to [Chapter 5, “Client / Server Communications.”](#)

Dynamic Scalable Architecture

Informix database servers implement an advanced architecture that Informix calls *Dynamic Scalable Architecture* (DSA). DSA provides distinct performance advantages for both single-processor and symmetric multiprocessor computers. These advantages, which this section describes further, are as follows:

- A small number of database server processes can service a large number of client application processes, with the following benefits:
 - Reduced operating-system overhead (fewer processes to run)
 - Reduced overall memory requirements
 - Reduced contention for resources within the database management system (DBMS)
- DSA provides more control over setting priorities and scheduling database tasks than the operating system does.

Parallel-Processing Architecture

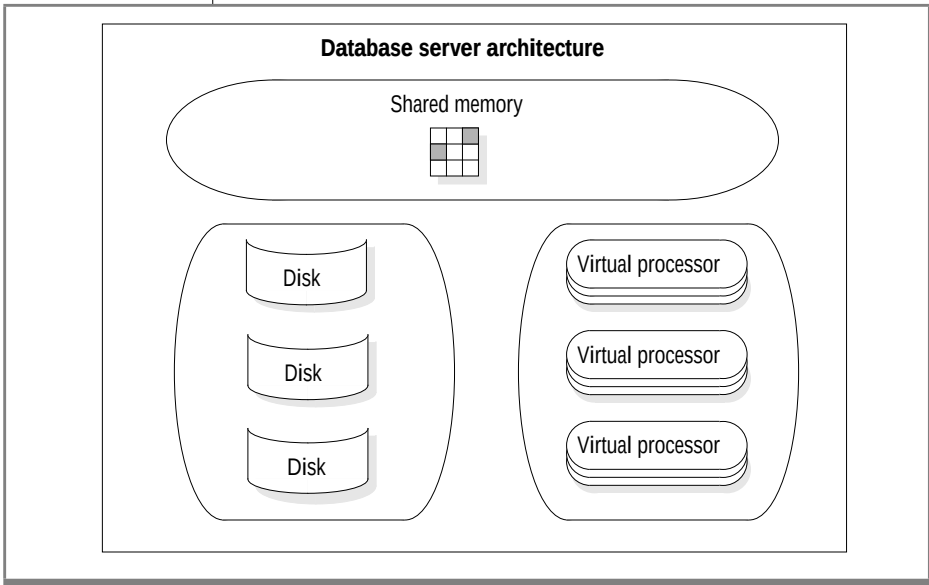
The multithreaded database server exploits *symmetric multiprocessor* (SMP) and uniprocessor architectures.

DSA particularly exploits SMP computer systems and provides the following benefits on those systems:

- Multiple processes can work in parallel for one client.
- On some multiprocessor computers, you can bind database server processes to specific CPUs.

Figure 1-1 shows the three major components of the Informix database server architecture.

Figure 1-1
Database Server
Architecture
Components



Informix database server architecture includes the following the three major components:

- Virtual processors

A 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. Database server virtual processors are *multithreaded* because they run multiple concurrent threads. For more information, refer to [Chapter 10, “Virtual Processors and Threads.”](#)

- Shared memory

Shared memory consists of a resident and virtual portion. The database server uses the resident portion to cache data from the disk for faster access by multiple client applications. The database server uses the virtual portion to maintain and control the resources that virtual processors require and to read in large blocks of infrequently accessed data. For more information on how the database server uses shared memory, see [Chapter 12, “Shared Memory,”](#) and [Chapter 13, “Managing Shared Memory.”](#)

- Disk component

The disk component is a collection of one or more units of disk space assigned to the database server. All the data in the databases and all the system information necessary to maintain the database server system reside within the disk component. For more information on the disk components, see [Chapter 14, “Data Storage.”](#)

The multithreaded database server manages access to one or more object relational databases for client applications. In an object relational database, data is organized in tables that consist of rows and columns.

Scalability

DSA allows the database server to scale its resources to the demands that applications place on it. A key element of DSA is the virtual processors that manage central processing, disk I/O, networking, and optical functions in parallel.

For more information on virtual processors, refer to [“Virtual Processors” on page 10-3](#). To understand how the database server manages shared memory to scale performance, see [Chapter 12, “Shared Memory.”](#) For tuning and performance information, refer to your [Performance Guide](#).

High Performance

The database server achieves high performance through the following mechanisms:

- Dynamic shared-memory management
- Raw disk management
- Dynamic thread allocation
- Parallelism

The following sections explain each of these mechanisms.

Dynamic Shared-Memory Management

All applications that use a single instance of a database server share data in the memory space of the database server. After one application reads data from a table, other applications can access whatever data is already in memory. This sharing of data in memory prevents redundant disk I/O and the corresponding degradation in performance that might otherwise occur.

Database server shared memory contains both data from the database and control information. Because the data needed by various applications is located in a single, shared portion of memory, all control information needed to manage access to that data can be located in the same place. The database server adds memory dynamically as needed, and as the administrator, you can also add segments to shared memory if necessary. For information about adding a segment to shared memory, refer to [Chapter 13, “Managing Shared Memory.”](#)

Direct Disk Access

The database server uses direct, or *unbuffered*, disk access to improve the speed and reliability of disk I/O operations. When you assign disk space to the database server, you can bypass the file-buffering mechanism that the operating system provides. The database server itself manages the data transfers between disk and memory.

UNIX provides unbuffered disk access by means of character-special devices (also known as *raw* disk devices). For more information about character-special devices, refer to your UNIX documentation. ♦

WIN NT

Windows NT provides unbuffered access through both NTFS files and raw disk devices. For more information about unbuffered NTFS files, refer to your Windows NT documentation. ♦

When you store tables on raw disks or unbuffered files, the database server can manage the physical organization of data and minimize disk I/O. When you store tables in this manner, you can receive the following performance advantages:

- The database server optimizes table access by guaranteeing that rows are stored contiguously.
- The database server bypasses operating-system I/O overhead by performing direct data transfers between disk and shared memory.

UNIX

If performance is not a primary concern, you can configure the database server to store data in regular (buffered) operating-system files, which are also known as *cooked* files. When the database server uses cooked files, it manages the file contents, but the operating system manages the disk I/O. ♦

WIN NT

On Windows NT, regular buffered files correspond to cooked files on UNIX. However, the database server automatically disables buffering for all of the NTFS files that it uses. Although the database server can use raw disk devices on Windows NT, Informix recommends that you use NTFS files to simplify disk administration. ♦

For more information about how the database server uses disk space, see [Chapter 14, “Data Storage.”](#)

Dynamic Thread Allocation

The database server supports multiple client applications using a relatively small number of processes called virtual processors. A virtual processor is a multithreaded process that can serve multiple clients and, where necessary, run multiple threads to work in parallel for a single query. In this way, the flexible database server architecture provides dynamic load balancing for both on-line transaction processing (OLTP) and decision-support applications.

WIN NT

On Windows NT, a virtual processor is implemented as a Windows NT thread rather than as a process. Therefore, both the operating system and the database server are multithreaded on Windows NT. Database server threads run within Windows NT threads.



Important: To keep the distinction clear between database server threads and Windows NT threads, this manual uses the term *process* to refer to both a UNIX process and a Windows NT thread. ♦

For a description of database server threads, refer to [Chapter 10, “Virtual Processors and Threads.”](#)

Fragmentation and Parallelism

The database server uses local table partitioning (also called *fragmentation*) to intelligently distribute tables across disks for better performance. For very large databases (VLDBs), the ability to fragment data is important to manage the data efficiently.

The database server can allocate multiple threads to work in parallel on a single query. This feature is known as parallel database query (PDQ).

PDQ is most effective when you use it with fragmentation. For an overview of fragmentation and PDQ, refer to [Chapter 16, “Table Fragmentation and PDQ.”](#)

Fault Tolerance

The database server uses the following logging and recovery mechanisms to protect data integrity and consistency in the event of an operating-system or media failure:

- Storage-space and logical-log backups
- Backup verification
- External backup and restore
- Fast recovery
- Point-in-time restore
- Restartable restore
- Mirroring
- Data replication

Storage Space and Logical-Log Backups of Transaction Records

The database server manages data in logical storage units called storage spaces. The most common storage space is the *dbspace*, in which the database server stores traditional data such as integer, decimal and floating-point numbers, fixed-length or variable-length character strings, and so forth. Other storage spaces include *blobspaces*, which store simple large objects, and *sbspaces*, which store smart large objects.

The database server stores transaction records and changes to the database server in *logical-log files*. You can back up storage spaces and logical-log files while users are accessing databases.

You can also create incremental backups on-line. Incremental backups enable you to back up only data that has changed since the last backup, which reduces the amount of time required to back up and restore your data and logical logs.

After a media failure, if critical data was not damaged (and the database server remains in on-line mode), you can restore only the data that was on the failed media, leaving other data available during the restore.

If the failure causes the database server to go off-line, you can restore all data, including the critical data and the root *dbspace*. An off-line restore is called a *cold restore*.

For more information on backup and restore, refer to one of the following manuals:

- *Backup and Restore Guide*
- [Archive and Backup Guide](#) (for ON-Archive on UNIX only and for **ontape**)

Backup Verification

If you use ON-Bar as your backup tool, you might want to verify the completeness and consistency of a storage-space backup. After you successfully verify a storage-space backup, you can restore it safely. If ON-Bar indicates problems with the backup, contact Informix Technical Support. The *Backup and Restore Guide* explains how to use ON-Bar to verify backups.

External Backup and Restore

An external backup allows you to make copies of disks that contain storage spaces without using ON-Bar. Later on, you can use an external restore to restore these disks to the database server without using ON-Bar or the Informix Storage Manager. External backups are especially useful if your site has special hardware or software that allows rapid copying of data directly to and from your primary data disks. The *Backup and Restore Guide* explains external backup and restore.

Fast Recovery

When the database server starts up, it checks if 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. During fast recovery, the database server uses its *logical log* and *physical log* to perform the following operations:

- Restore the databases to their state at the last checkpoint
- Roll forward all committed transactions since the last checkpoint
- Roll back any uncommitted transactions

The database server spawns multiple threads to work in parallel during fast recovery. For a detailed explanation of fast recovery, refer to [Chapter 23, “Checkpoints and Fast Recovery.”](#)

Point-in-Time Restore

After the time of the backup, you can restore data from backup media to a specified point in time. This feature enables you to restore a corrupted database to a point at which you know that the data was reliable. For more information, refer to the *Backup and Restore Guide* or the [Archive and Backup Guide](#).

Restartable Restore

If something goes wrong during an ON-Bar restore, you can restart it at the point that it failed. You do not have to restart the restore from the beginning. The *Backup and Restore Guide* explains restartable restore.

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.

The database server relies on the operating system for bad-sector mapping. When the database server confirms the failure of a disk, it suspends I/O operations on that chunk. If the chunk has been mirrored, the database server directs I/O requests to the mirror. If an unmirrored chunk containing critical information fails, the database server shuts down immediately. Critical information includes logical-log files, the physical log, and the root dbspace. The chunks on which this data resides are referred to as *critical dbspaces*.



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

For more information about mirroring and critical data, refer to [Chapter 24, “Mirroring.”](#)

High-Availability Data Replication

If your organization requires a high degree of availability, you can replicate the database server and its databases and run them simultaneously on a second computer. Replicating the database server and its databases provides a backup system in the event of a catastrophic failure; if one site experiences a disaster, applications can immediately use the second database server in the pair. For more information about data replication, refer to [Chapter 27, “High-Availability Data Replication.”](#)

Running data replication also allows you to balance read-only applications (for example, decision-support applications) across both database servers in the data-replication pair.

UNIX

Dynamic System Administration

Distributed databases require dynamic system administration tools to monitor and tune the database server. The database server provides many administration tools to help you perform these tasks easily. These tools include the system monitoring interface (SMI), command-line utilities, and graphical administration tools.

The database server provides the **onperf** performance utility, a graphical administration tool. The **onperf** utility displays database server activity with the Motif window manager. The **onperf** utility allows you to monitor most of the same database server metrics as the **onstat** utility reports. For more information about **onperf**, refer to your [Performance Guide](#). ♦

Database Server Security

The database server enforces access privileges to databases and tables through the use of the SQL statements GRANT and REVOKE. For more information about database and table privileges, refer to the [Informix Guide to Database Design and Implementation](#) and the [Informix Guide to SQL: Syntax](#).

Year 2000 Compliance

The database server and client products support the **DBCENTURY** environment variable for Year 2000 compliance. The **DBCENTURY** environment variable lets you choose the appropriate expansion (present, past, closest, or future century) for DATE and DATETIME values that have only a one- or two-digit year. For more information, see the [Informix Guide to SQL: Reference](#).

UNIX

Auditing

In addition to database security, the database server offers the ability to audit database events on a database server-wide basis. Auditing, described in your [Trusted Facility Manual](#), enables you to track which users performed which actions to which objects at what time. You can use this information to monitor database activity for suspicious use, deter unscrupulous users, or even serve as evidence of database server abuse.

Distributed Database Queries

The database server allows users to query (and update) more than one database across multiple database servers within a single transaction. The database servers can reside on a single host computer or on the same network. For a list of the types of networks that the database server supports, refer to [Chapter 5, “Client/Server Communications.”](#)

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 Informix Enterprise Gateway products to manipulate data in non-Informix databases. 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 more information on these protocols, refer to [Chapter 30, “Multiphase Commit Protocols.”](#)

UNIX

You can also use the database server in a heterogeneous environment that conforms to X/Open. For more information about using the database server within an X/Open environment, refer to your [TP/XA Programmer's Manual](#). ♦

Extensibility

Dynamic Server enables you to extend the capabilities of the database server by adding user-defined data types and user-defined routines to support them. The combination of a new data type and supporting routines is called a DataBlade module. You can purchase DataBlade modules, from Informix or third-party vendors, to support many such data types.

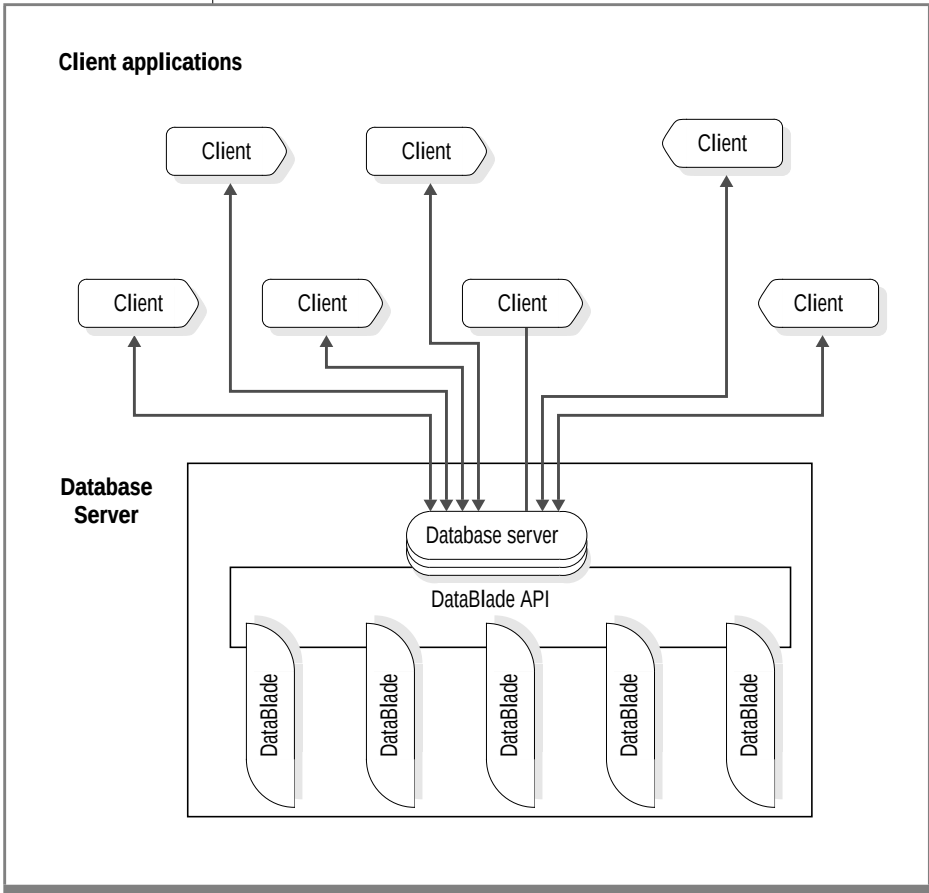
The database server supports the following types of extensibility:

- User-defined data types
- Storage and access of simple and smart large objects
- User-defined routines
- DataBlade modules
- User-defined virtual processors
- User-defined access methods

For more information on the different data types, refer to the [Informix Guide to SQL: Reference](#). For more information on how to create new data types, refer to [Extending Informix Dynamic Server 2000](#).

Figure 1-2 illustrates the extensibility that DataBlade modules add to the database server.

Figure 1-2
Database Server
with DataBlade
Module Extensions



The following sections describe in more detail the support that the database server provides for new data types.

New Data Types

Relational database management systems traditionally have supported alphanumeric data such as fixed-length or variable-length character strings, integers, decimal and floating-point numbers, date, time, and time-interval structures. The database server provides *built-in* support for these data types.

Dynamic Server is a database server in an *object-relational database-management system* that provides object-oriented capabilities that enable you to *extend* its data management capabilities to add the following new data types:

- User-defined distinct data types
- User-defined opaque data types and the capability to extend the following data types using opaque data types:
 - Complex data types
 - Type and table inheritance
 - User-defined casting of opaque data types

The database server also inherently supports complex data types and allows you to define and implement new data types. As with built-in data types, you perform the following administrative tasks:

- Define tables with columns containing these new data types
- Define storage spaces for these new data types
- Manage the storage spaces
- Create user-defined routines (triggers, functions, and so forth) that use these new data types as arguments

For information about the built-in and extended data types, refer to the [*Informix Guide to SQL: Reference*](#).

User-Defined Data Types

The database server also enables you to develop your own user-defined routines and access methods to support new data types. User-defined data types can be opaque or distinct.

An *opaque type* stores a single value and cannot be divided into components by the database server. It is implemented as a structure and a set of routines that allow the database server to support the data type. Routines that support a new opaque type pass the contents of the structure to the database server to store in the database.

A *distinct type* has the same storage representation as an existing opaque type but is distinguished by its name and cannot be substituted for the source data type. For example, you can create a new distinct type **decnum** with the same representation as **real**. All routines that operate on **real** values have database server-simulated counterparts for **decnum** values. However, **decnum** values and **real** values cannot be added, subtracted, or compared with one another without explicitly *casting* one value to the type of the other. A cast function converts one data type to another data type.

For information on implementing new data types, refer to [Extending Informix Dynamic Server 2000](#). For an introduction to user-defined routines, see “User-Defined Routines and DataBlade Modules” on page 1-24.

Complex Data Types

The database server enables you to build complex data types from a combination of built-in, user-defined, or other complex data types. In contrast to user-defined *opaque types*, whose characteristics are not known to the database server, you can use SQL statements to access the components of a complex data type.

For more information on complex data types, see the chapter on complex data types in the [Informix Guide to Database Design and Implementation](#).

Data Type and Table Inheritance

The database server supports *inheritance*, which allows named row types and typed tables to inherit representation and behavior from parent types and tables. A *named row type* is a group of fields that are defined under a single name.

The inherited representation affects how you define and manage the storage spaces for these row types and typed tables. The inherited behavior facilitates definition of these typed tables.

For more information on inheritance, see the chapter on complex data types in the [Informix Guide to Database Design and Implementation](#).

Support for Large Objects

Relational database systems have provided the ability to store large chunks of non-alphanumeric data as binary or text data types. Client applications, however, were left with the problem of accessing and manipulating the content of these objects after the database server retrieved them.

Dynamic Server can use *simple large objects* and *smart large objects* to store large data objects in a database:

- Simple large objects store TEXT and BYTE data.
- Smart large objects store CLOB (character large object) and BLOB (binary large object) data. Smart large objects provide the ability to seek, read from and write to segments of the object. Smart large objects are also recoverable and obey transaction isolation modes.

Simple Large Objects and Blobspaces

You can store simple large objects in either the same dbspace as the table or in a separate *blobspace*. A blobspace is a collection of database server storage spaces that exclusively store simple large objects. For information on blobspaces, see [“Blobspaces” on page 14-24](#). For information on how to create a blobspace, see the information on **onspaces** in the utilities chapter of the *Administrator's Reference*.

Smart Large Objects and Sbspaces

You can use smart large objects to store user-defined data types such as video and audio clips, pictures, large text documents, and spatial objects such as drawings and maps. The database server stores large objects in *sbspaces*. Sbspaces are collections of database server storage spaces that exclusively store smart large objects. For information on sbspaces, see “[Sbspaces](#)” on [page 14-24](#). For information on how to create an sbspace, see the information on **onspaces** in the utilities chapter of the *Administrator's Reference*.

E/C

Informix ESQL/C provides client applications with the Smart Large Object application programming interface (API), which enables an application to access a smart large object in much the same way as an operating-system file. This API enables you to access any portion of a smart large object. For information on how to access either a simple large object or a smart large object from a client application, refer to the *Informix ESQL/C Programmer's Manual*. ♦

DB API

An application can also store or retrieve smart large objects in pieces using the DataBlade API. For more information on the DataBlade API functions, refer to the *DataBlade API Programmer's Manual*. ♦

Support for External Spaces

The database server supports *extspaces* (*external spaces*), which are storage spaces that the database server does not manage directly. An extspace is a description of an external location. It provides a way to associate a logical name with a location string such as a pathname or network gateway, for example.

You can specify an external space as the storage space for a table or index for which you create a user-defined access method. For an introduction to user-defined access methods, refer to “[User-Defined Access Methods](#)” on [page 1-25](#).

For more information on extspaces, refer to “[Extspaces](#)” on [page 14-35](#). For information on creating and dropping extspaces, refer to the information on **onspaces** in the *Administrator's Reference*. For information on specifying an external space as the storage location for a table or index, see the CREATE TABLE and CREATE INDEX statements in the *Informix Guide to SQL: Syntax*.

User-Defined Routines and DataBlade Modules

You can extend data management capabilities with one of the following methods:

- Develop your own user-defined routines to use on existing data types.

You can write these user-defined routines in, Stored Procedure Language (SPL), C, or Java programming language. For information on implementing user-defined routines, see [Extending Informix Dynamic Server 2000](#).

- Add a DataBlade *module*.

A DataBlade *module* is a pre-packaged user-defined data type and all the required supporting routines. You can obtain DataBlade modules from Informix and from third-party vendors.

- Use the database server features that enable you to add data types, supporting routines, and access methods.

To create your own DataBlade module, you can use one of the following methods:

- DataBlade Developers Kit

For more information, refer to *DataBlade Developers Kit User's Guide*.

- DataBlade API functions

For more information, refer to [DataBlade API Programmer's Manual](#).

User-Defined Virtual Processors

To isolate the effects of user-defined routines that your programmers might develop, you can define a separate virtual-processor (VP) class in which to execute these user-defined routines. For general information on virtual processors, see [Chapter 10, "Virtual Processors and Threads."](#) For more information on the use of user-defined virtual processors to execute user-defined routines, see [Extending Informix Dynamic Server 2000](#).

User-Defined Access Methods

In addition to adding user-defined data types and user-defined routines, you can also create user-defined access methods to support new data types or to include in a database information that is stored externally to the database server.

An *access method* is a set of database server functions that a database server uses to access and manipulate a table or an index. Dynamic Server supports the following types of access methods:

- Primary access methods

A *primary access method* is a set of database server functions that access and manipulate data. It is a set of routines that perform operations such as inserting, deleting, updating, and scanning data in a table. Dynamic Server provides all of the necessary routines for accessing the built-in data types.

- Secondary access methods

A *secondary access method* is a set of routines that implement an index.

For information about installing and using the access method, refer to your DataBlade user guide or access-method user guide. For information about using the access method with the CREATE TABLE (primary access method) or CREATE INDEX (secondary access method) statements, refer to the [Informix Guide to SQL: Syntax](#).

Overview of Database Server Administration

Database Server Administrator.	2-3
Initial Tasks	2-4
Configuration Tasks	2-4
Managing Disk Space	2-4
Managing Database-Logging Status.	2-5
Managing the Logical Log	2-5
Managing the Physical Log.	2-5
Using Auditing.	2-5
Using Mirroring	2-6
Using Data Replication	2-6
Managing Shared Memory	2-6
Managing Virtual Processors	2-7
Managing Parallel Database Query	2-7
Routine Tasks.	2-7
Changing Modes	2-7
Backing Up Data and Logical-Log Files	2-8
Monitoring Database Server Activity	2-8
Checking for Consistency	2-8
Summary of Administration Tasks	2-9
Monitoring Database Server Activity	2-12
Sources of Information for Monitoring	2-13
Message Log.	2-13
Monitoring the Message Log.	2-14
Changing the Destination for Message-Log Messages	2-14
Event Alarm.	2-14

System Console	2-15
Windows NT Event Viewer.	2-15
ON-Monitor	2-15
SMI Tables	2-15
onstat Utility.	2-16
oncheck Utility	2-17
onperf Tool	2-17

In This Chapter

As a database server administrator, you need to be aware of the tasks and responsibilities that fall into your domain. This chapter describes the three types of tasks that you need to perform as an administrator of an Informix database server:

- Initial installation and configuration tasks
- Routine tasks that you perform on a regular basis
- Configuration tasks that you perform less frequently

For a summary of database server administration tasks, refer to [“Summary of Administration Tasks” on page 2-8](#).

Database Server Administrator

Most database server administrative tasks require you to have the privileges accorded to the database server administrator. On UNIX, you must log in as user **informix** to acquire the privileges of the database server administrator. On Windows NT, you must belong to the **Informix-Admin** group to administer the database server.

Initial Tasks

When you first acquire the database server, you need to perform some initial installation and configuration tasks. For a description of these tasks, see [Chapter 3, “Installing and Configuring the Database Server.”](#)

If you are moving from one version of the database server to another, refer to the [Informix Migration Guide](#).

You must also configure connectivity for your database server and client applications, as explained in [Chapter 5, “Client/Server Communications.”](#)

These tasks can seem complicated and time consuming. Fortunately, they are not common tasks and are not representative of most of the administrative work that the database server requires.

Configuration Tasks

Configuration tasks are generally either setup tasks that involve initiating functionality or maintenance and performance-adjustment tasks that are required by the usage pattern of your database server.

Managing Disk Space

You are responsible for planning and implementing the layout of information managed by the database server on disks. The way you distribute the data can greatly affect the performance of the database server.

[Chapter 14, “Data Storage,”](#) explains the advantages and drawbacks of different disk configurations. [Chapter 15, “Managing Disk Space,”](#) describes the actual disk-management tasks.

If you plan to use more than one database server instance on the same computer, refer to [Chapter 6, “Multiple Residency,”](#) to familiarize yourself with the issues of multiple residency.

For information on how to fragment tables and indexes over multiple disks to improve performance, concurrency, data availability, and backups, refer to [Chapter 16, “Table Fragmentation and PDQ,”](#) and to your [Performance Guide](#).

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.

For information about these logging options, refer to [Chapter 17, “Logging.”](#)
For information on how to change logging options, refer to [Chapter 18, “Managing Database-Logging Status.”](#)

Managing the Logical Log

Although backing up logical-log files is a routine task, logical-log administration (placing and sizing log files and specifying high-water marks) is required even when none of your databases use transaction logging. For information about logical-log administration, refer to [Chapter 19, “Logical Log.”](#)

For instructions on creating and modifying the logical-log configuration, see [Chapter 20, “Managing Logical-Log Files.”](#)

For information on backing up the logical log, see the *Backup and Restore Guide* and the [Archive and Backup Guide](#).

Managing the Physical Log

You can change the size and location of the physical log as part of effective disk management. For more information about the physical log, refer to [Chapter 22, “Managing the Physical Log.”](#)

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 the [Trusted Facility Manual](#).

Using Mirroring

For information on mirroring, refer to [Chapter 24, “Mirroring.”](#) Informix recommends that you mirror at least your root dbspace. For instructions on tasks related to mirroring, see [Chapter 25, “Using Mirroring.”](#)

Using Data Replication

Informix database servers supports high-availability data replication and Informix Enterprise Replication. High-availability data replication replicates an entire database to another database server, providing a hot standby in case of a catastrophic computer failure. Enterprise Replication supports asynchronous data replication and allows you to select the exact data you require for replication. For more information on high-availability data replication, see [Chapter 27, “High-Availability Data Replication,”](#) and [Chapter 28, “Using High-Availability Data Replication.”](#) For information on Enterprise Replication, see the *Guide to Informix Enterprise Replication*.

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

For information on how the database server uses shared memory, refer to [Chapter 12, “Shared Memory.”](#) For a description of how to manage shared memory, refer to [Chapter 13, “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 10, “Virtual Processors and Threads.”](#) For information on the procedures to manage virtual processors, refer to [Chapter 11, “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 on-line 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](#).

Routine 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 off-line mode, so database server operating mode changes would not be a routine task.

Changing Modes

The database server administrator is responsible for starting up and shutting down the database server by changing the mode. [Chapter 8, “Managing Database Server Operating Modes,”](#) 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, Informix recommends that you make frequent backups of your storage spaces and logical logs.

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 or logical log 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, see the *Backup and Restore Guide*. For information on using **ontape** or ON-Archive, see the [Archive and Backup Guide](#).

Monitoring Database Server Activity

The Informix database server design lets you monitor every aspect of the database server. “[Sources of Information for Monitoring](#)” on page 2-13 provides descriptions of the available information, instructions for obtaining information, and suggestions for its use. As a result of monitoring, you might need to change your configuration in one of the ways described in “[Summary of Administration Tasks](#)” on page 2-8.

Checking for Consistency

Informix recommends that you perform occasional checks for data consistency. For a description of these tasks, refer to [Chapter 29, “Consistency Checking.”](#)

Summary of Administration Tasks

The following tables summarizes the initial tasks, routine tasks, and configuration tasks that you perform to administer the database server. [Figure 2-1](#) lists initial administration tasks.

Figure 2-1
Initial Tasks Summary

Initial Tasks	Reference
Plan for database server	“Planning for the Database Server” on page 3-4
Preinstallation operating-system changes	“Configuring the Operating System” on page 3-6
Install the database server	<i>Installation Guide</i>
Configure the database server	Chapter 4, “Configuration Parameters”
Define connectivity	Chapter 5, “Client / Server Communications”
Initialize the database server	Chapter 9, “Initializing the Database Server”

Figure 2-2 lists database server configuration tasks.

Figure 2-2
Configuration Tasks Summary

Configuration Tasks	Reference
Manage disk space and storage spaces	Chapter 15, “Managing Disk Space”
Analyze disk configuration	Chapter 14, “Data Storage”
Changing database-logging status	Chapter 17, “Logging” Chapter 18, “Managing Database-Logging Status”
Administer the logical log: ■ Size and placement of logical logs ■ Modify logical log configuration	Chapter 19, “Logical Log” Chapter 20, “Managing Logical-Log Files”
Administer the physical log	Chapter 22, “Managing the Physical Log” Chapter 23, “Checkpoints and Fast Recovery”
Administer auditing	Trusted Facility Manual
Manage mirroring	Chapter 24, “Mirroring” Chapter 25, “Using Mirroring”
Replicate data	Chapter 27, “High-Availability Data Replication” Chapter 28, “Using High-Availability Data Replication” Guide to Informix Enterprise Replication
Manage shared memory	Chapter 12, “Shared Memory” Chapter 13, “Managing Shared Memory”
Manage virtual processors	Chapter 11, “Managing Virtual Processors”
Manage parallel data query (PDQ)	Performance Guide
Migrate to a different database server version	Informix Migration Guide

Figure 2-3 lists routine administration tasks.

Figure 2-3
Routine Tasks Summary

Routine Tasks	Reference
Change database server modes	Chapter 8, “Managing Database Server Operating Modes”
Refresh data warehousing tables	<i>Administrator’s Reference</i>
Back up data and logical-log files:	
■ ON-Archive (UNIX)	Archive and Backup Guide
■ ontape utility	Archive and Backup Guide
■ ON-Bar	<i>Backup and Restore Guide</i>
■ Informix Storage Manager	Informix Storage Manager Administrator’s Guide
Verify storage-space backup	<i>Backup and Restore Guide</i>
Perform external backups and restores	<i>Backup and Restore Guide</i>
	For information about using the onmode -c commands to block or unblock the database server, see the <i>Administrator’s Reference</i> .
Monitor activity	Monitoring information in relevant chapters of this book
Check for data consistency	Chapter 29, “Consistency Checking”
Tune database server performance	Performance Guide
Work with user-defined routines and extended data types	Extending Informix Dynamic Server 2000

Monitoring Database Server Activity

You can monitor information about the following aspects of the database server:

- Configuration
- Checkpoints
- Shared memory
 - Shared-memory segments
 - Shared-memory profile
 - Buffers
 - Latches
 - Locks
- Active tablespaces
- Virtual processors
- Sessions and threads
- Transactions
- Parallel database queries (PDQ) and resources
- Databases
- Logging activity
 - Logical-log files
 - Physical-log file
 - Physical-log and logical-log buffers
 - Log backup status
- Disk usage
 - Chunks
 - Tablespaces and extents
 - Simple large objects in dbspaces
 - Simple large objects in blobspaces
- High-availability data-replication information

Sources of Information for Monitoring

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

Source of Information	UNIX	Windows NT
Message log	✓	✓
Event alarm	✓	✓
System console	✓	
Windows NT Event Viewer		✓
ON-Monitor	✓	
SMI tables	✓	✓
onstat utility	✓	✓
oncheck utility	✓	✓
onperf utility	✓	

The following sections explain each of these sources.

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 Alarm” on page 2-14](#). To specify the message-log pathname, set the MSGPATH configuration parameter. For more information about MSGPATH, see the chapter on configuration parameters in the *Administrator’s Reference*.

Monitoring the Message Log

Informix recommends 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. For a list of these messages, see the chapter on message-log messages in the *Administrator's Reference*.

Monitor the message-log size as well 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.

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, Informix does not expect you to perform this kind of monitoring.

Changing the Destination for Message-Log Messages

You can change the value of MSGPATH while the database server is in on-line mode, but the changes do not take effect until you reinitialize shared memory.

Event Alarm

The database server provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server environment. This mechanism is the event-alarm feature.

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*.

UNIX

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*.

You can change the value of `CONSOLE` while the database server is in on-line mode, but the changes do not take effect until you reinitialize shared memory.

WIN NT

Windows NT 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, choose **Log→ Security**. Then double-click any event for a more detailed message.

UNIX

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 chapter about ON-Monitor in the *Administrator's Reference*.

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*.

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, and users.

One useful feature of **onstat** output is the heading that indicates the database server status. Whenever the database server is blocked, **onstat** displays the following line after the banner line:

```
Blocked: reason
```

The variable *reason* can take one of the following values.

Reason	Description
CKPT	Checkpoint
LONGTX	Long transaction
ARCHIVE	Ongoing archive
MEDIA_FAILURE	Media failure
HANG_SYSTEM	Database server failure
DBS_DROP	Dropping a dbspace
DDR	Discrete high-availability data replication
LBU	Logs full high-water mark

For an example of what **onstat** displays when the database server is blocked to preserve logical-log space for administrative tasks, see [“Monitoring the Logical Log for Fullness” on page 20-16](#).

UNIX

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*.

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](#).

Configuration

- Chapter 3** **Installing and Configuring the Database Server**
- Chapter 4** **Configuration Parameters**
- Chapter 5** **Client/Server Communications**
- Chapter 6** **Multiple Residency**
- Chapter 7** **Using Multiple Residency**

Section II



Installing and Configuring the Database Server

In This Chapter	3-3
Planning for the Database Server	3-4
Considering Your Priorities	3-4
Considering Your Environment	3-5
Configuring the Operating System	3-6
Configuring Windows NT	3-6
Providing NTFS Partitions	3-6
Configuring Windows NT Memory	3-6
Modifying UNIX Kernel Parameters	3-7
Allocating Disk Space	3-7
Creating a Raw Device or Unbuffered File	3-8
Creating Standard Device Names	3-8
Setting Permissions, Ownership, and Group.	3-9
Installing the Database Server	3-9
Using Multiple Residency	3-9
Upgrading the Database Server	3-10
Creating a Private Installation.	3-10
Setting Environment Variables	3-10
Required Environment Variables.	3-11
Global Language Support	3-12
Other Environment Variables	3-12
Environment Variable Files.	3-13
Configuring Connectivity	3-13
The sqlhosts File	3-14
Connectivity Information on Windows NT	3-14

Preparing the ONCONFIG Configuration File	3-14
Creating a Configuration File	3-15
Creating an ONCONFIG File on UNIX	3-15
Creating an ONCONFIG File on Windows NT.	3-16
Starting and Administering the Database Server.	3-17
Preparing to Connect to Applications	3-17
Starting the Database Server and Initialize Disk Space	3-17
Performing Administrative Tasks	3-18
Preparing for Automatic Startup on Windows NT	3-19
Preparing the UNIX Startup and Shutdown Scripts	3-19
Making Sure That Users Have the Correct Environment Variables	3-21
Warning UNIX System Administrator About cron Jobs.	3-21
Setting Up Your Storage Manager and Storage Devices	3-22
Creating Storage Spaces	3-22
Monitoring Configuration Information	3-23
Using Command-Line Utilities	3-23
onstat -c	3-23
oncheck -pr	3-23
Using ON-Monitor.	3-24

In This Chapter

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 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, a database server instance that serves 1000 users who execute frequent, short transactions is much different from the database server instance in which a few users make long and complicated searches.

This chapter has three purposes:

- To introduce files, environment variables, and utilities that the database server uses for configuration
- To let you quickly start the database server with a simple configuration
- To provide a foundation for more detailed information that appears throughout this book

The chapter also introduces terminology and covers some of the issues that you must consider before you install the database server. The following topics appear in this chapter:

- Planning for the database server
- Configuring the operating system
- Allocating disk space
- Installing the database server
- Setting environment variables
- Configuring connectivity information
- Preparing the ONCONFIG configuration file

- Starting and administering the database server
- Monitoring configuration information

If you are installing one or more DataBlade modules, install them after you have installed the database server. For instructions on installing the DataBlade module, refer to the DataBlade module documentation.

Planning for the Database Server

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?
- How much help or supervision will the users require?
- 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?
- How much does the database server instance need to do without supervision?
- Does the database server usually handle many short transactions or fewer long transactions?

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 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 [Informix Storage Manager Administrator's Guide](#) or your storage-manager documentation.
- 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

WIN NT

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.

Configuring Windows NT

On Windows NT, you must create NTFS partitions and configure memory.

Providing NTFS Partitions

All dbspaces must be stored in a New Technology File System (NTFS). If all of your partitions are File Allocation Table (FAT) partitions, you need to convert at least one partition to NTFS. You can use the Windows NT **convert** utility, as the following example shows:

```
convert /fs:ntfs
```

A database server system administrator can log in to the console from any node to perform system management and monitoring tasks. For details on how to set up your system console, refer to your operating-system administration guide.

Configuring Windows NT Memory

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 that you use to configure 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 13, “Managing Shared Memory.”](#)

Allocating Disk Space

Configuring your disks is the most important task for obtaining 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.

To allocate disks for the database server

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

Creating a Raw Device or Unbuffered File

To achieve better performance, UNIX uses raw disk devices, and Windows NT uses unbuffered NTFS files to bypass the buffering of disk I/O that the operating system normally performs. To create raw devices on UNIX, follow the instructions provided with your operating system. To configure unbuffered NTFS files on Windows NT, follow the steps in the installation program.

Some operating systems use the concept of a *logical volume*, and others use a *logical unit*. Each of these terms represents the basic unit of physical disk space that the operating system reserves for the database server. A chunk is a physical partition, logical volume, a logical unit, or regular file that has been assigned to the database server. Use the **onspaces** utility to create chunks.

Important: You must define your logical volume or logical unit to be no larger than a chunk. To determine which chunk size your operating system supports, refer to your machine notes file.



UNIX

Creating Standard Device Names

Informix recommends 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** (**ls -lg** on BSD) on your device directory to verify that both the devices and the links exist. The following example shows links to raw devices. If your operating system does not support symbolic links, you can use hard links.

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

Setting Permissions, Ownership, and Group

Files or raw 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).

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 your UNIX documentation. ♦

WIN NT

On Windows NT, files must be owned by a member of the **Informix-Admin** group. ♦

Installing the Database Server

Installation refers to the process of loading files from the product distribution media onto your computer and running the installation script to set up the product files correctly. For information about how to install the database server, see your [Installation Guide](#).



Warning: Do not try to install multiple copies of the database server from the distribution media on the same computer.

Using Multiple Residency

If you run more than one instance of the database server on the same computer, it is called *multiple residency*. To prepare to use multiple instances of the database server, first install and configure *one* database server. To prepare for multiple residency, initialize multiple instances of the database server. For more information, refer to [Chapter 7, “Using Multiple Residency.”](#)

UNIX

Upgrading the Database Server

If you are upgrading your database server from an earlier version, see the [Informix Migration Guide](#) for instructions.

Creating a Private Installation

A private installation provides support for programmers who develop DataBlade modules and user-defined routines. It allows a developer to attach a debugger and test code extensions to the database server without affecting the performance of the database server or the work of other users.

A private installation of the database server allows a developer to create new functions, dynamically link them to the database server, execute them, and debug them without violating security. In a private installation of the database server, you (and not user **informix**) are the owner of critical resources and services such as the **sysmaster** database.

To create a private installation, use the **installserver** script. For more information, see your [Installation Guide](#).

WIN NT

Setting Environment Variables

To start, stop, or access a database server, you must set the appropriate environment variables.

On Windows NT, 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. ♦

UNIX

On UNIX, you can include the environment variables in the operating-system boot-up procedure, include the environment variables in your **.cshrc** or **.informix** file, or set the environment variables each time you log on. For information about **.cshrc**, refer to your operating-system manuals. For information about the **.informix** file and others files that the database server uses, refer to the *Administrator's Reference*. ♦



Tip: The “*Informix Guide to SQL: Reference*” contains a complete list of environment variables.

Required Environment Variables

You must set the following environment variables before you access the database server or perform most administrative tasks:

■ INFORMIXDIR

The **INFORMIXDIR** environment variable specifies the directory where the product files are installed. Set **INFORMIXDIR** to the directory where you installed your Informix database server.

■ PATH

The **PATH** environment variable specifies the location of executable files.

Operating System	Directory
UNIX	\$INFORMIXDIR/bin
Windows NT	%INFORMIXDIR%\bin

■ ONCONFIG

The **ONCONFIG** environment variable specifies the name of the active ONCONFIG configuration file. The next section describes how to prepare the ONCONFIG file. After you prepare the ONCONFIG file, set the **ONCONFIG** environment variable to the name of the file.

Users running client applications do not need to set the **ONCONFIG** environment variable.

If the **ONCONFIG** environment variable is not present, the database server uses configuration values from the following files.

Operating System	ONCONFIG File
UNIX	\$INFORMIXDIR/etc/onconfig
Windows NT	%INFORMIXDIR%\etc\onconfig.std

GLS

■ **INFORMIXSERVER**

The **INFORMIXSERVER** environment variable specifies the name of the default database server. It has the value specified for the **DBSERVERNAME** or **DBSERVERALIASES** configuration parameter in the **ONCONFIG** file.

Global Language Support

The GLS feature allows you to create databases that use the diacritics, collating sequence, and monetary and time conventions of the language that you select. To use GLS, you must set appropriate environment variables. Users might need additional environment variables, such as **LC_COLLATE**, to describe their environment fully. For a detailed discussion of GLS, refer to the [Informix Guide to GLS Functionality](#).

Other Environment Variables

The **DBSPACETEMP** environment variable specifies dbspaces that the database server can use to store temporary tables for a particular session. Using **DBSPACETEMP** can improve performance.

DBSPACETEMP specifies the location of dbspaces. If **DBSPACETEMP** is not set, the default location is the root dbspace.

For further information about **DBSPACETEMP**, refer to the chapter about configuration parameters in the *Administrator's Reference*. For performance considerations, refer to your [Performance Guide](#).

UNIX

The **TERM**, **TERMCAP**, **TERMINFO** and **INFORMIXTERM** environment variables specify the type of terminal interface. The basic login procedure usually sets the **TERM** environment variable. This environment variable is not required for initialization, but it must be set before you can run an application. The other three environment variables might be required, depending on your environment. You might need assistance from the UNIX system administrator to set these variables because they are highly system dependent. ♦

Environment Variable Files

To set additional environment variables for each database server user, the database administrator can prepare an environment-configuration file.

Operating System	Directory
UNIX	<code>\$INFORMIXDIR/etc/informix.rc</code>
Windows NT	<code>%INFORMIXDIR%\etc\informix.rc</code>

UNIX

To override environment variables that have been automatically set, users can use a private environment-variable file, `~/.informix`, or assign new values to environment variables individually. ♦

WIN NT

To override environment variables that have been automatically set, set them in the **System→Environment→User Variables** control application. ♦

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 is known.

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

For detailed information about the connectivity information, see [Chapter 5, “Client / Server Communications.”](#)

UNIX

The sqlhosts File

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.

WIN NT

Connectivity Information on Windows NT

On Windows NT, the HKEY_LOCAL_MACHINE registry contains the connectivity information. Informix suggests that you *not* edit the HKEY_LOCAL_MACHINE registry. The database server installation procedure prepares the registry information. The **setnet32** utility lets you manage host information. For information about **setnet32**, see the installation information in your client documentation.

For more information about preparing the SQLHOSTS connectivity information, refer to the documentation notes described in [“Documentation Notes, Release Notes, Machine Notes”](#) on page 14 of the Introduction.

Preparing the ONCONFIG Configuration File

After the database server is installed, it must be configured before it can be brought on-line. *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.

Defaults for most of the configuration parameters are set during the installation of the product. However, you can change the configuration parameters to customize the database server for your data-processing environment.

[Chapter 4, “Configuration Parameters,”](#) provides an overview of the configuration parameters. For information about additional configuration parameters, see the chapter on configuration parameters in the *Administrator’s Reference*.

The configuration parameters customize the database server for your data-processing environment. This manual refers to the file that stores the configuration parameters as the *ONCONFIG file*. You must prepare this file before you initialize the database server. After you prepare the configuration file, you must set the **ONCONFIG** environment variable to identify it to the database server.

Creating a Configuration File

Use the **onconfig.std** template file for your configuration file. For a listing of this file, see the appendix on database server files in the *Administrator's Reference*. For more information on the configuration parameters listed in **onconfig.std**, refer to [Chapter 4, "Configuration Parameters."](#)



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

Make a copy of the appropriate template file and set the **ONCONFIG** environment variable to refer to your copy.



Important: 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.

For the default values contained in the **onconfig.std**, refer to the configuration parameter chapter 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 9-6](#).

UNIX

Creating an ONCONFIG File on UNIX

To prepare the ONCONFIG file, use a text editor and complete the following steps:

1. Make a copy of a standard ONCONFIG file template.

```
$ INFORMIXDIR/etc/onconfig.std
```

Store the new file in the same directory as the template. You can give your new configuration file any name that meets the requirements of your operating system.

WIN NT

2. Edit your new ONCONFIG file to modify the configuration parameters that you decide to change.
3. Set the **ONCONFIG** environment variable to the name of your new ONCONFIG file.

Creating an ONCONFIG File on Windows NT

In Windows NT, a new instance of the database server is created and initialized when you install the Informix software. You can also 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, refer to [“Creating a New Database Server on Windows NT” on page 7-5](#). You can also use a text editor to create a new configuration file.

To use a text editor

1. Copy the following file to **%INFORMIXDIR%\etc\%ONCONFIG%:**
`%INFORMIXDIR%\etc\onconfig.std`
2. Edit the new file and change the configuration parameters.
3. Use the **regedt32** tool to change your **ONCONFIG** environment variable to the name of your new ONCONFIG file.

To use the regedt32 tool to change the ONCONFIG environment variable

1. Start the **regedt32** tool.
2. Open the following folder (replace *DBSERVERNAME* with the name of the database server):

```
HKEY_LOCAL_MACHINES\SOFTWARE\Informix\OnLine  
  \DBSERVERNAME\Environment
```

3. Edit the ONCONFIG value.

For existing variables, double-click the variable value and change the value in the String Editor window.

Starting and Administering the Database Server

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

- Prepare to connect to applications.
- Start the database server and initialize disk space.
- Create dbspaces.
- Perform administrative tasks.

Preparing to Connect to Applications

Application programs use the **sqlhosts** file or registry to look up connectivity information. On UNIX, the **sqlhosts** file resides in the **\$INFORMIXDIR/etc** directory. On Windows NT, client applications use the **sqlhosts** registry. For more information about the **sqlhosts** file and registry, refer to [“The sqlhosts File and the SQLHOSTS Registry Key” on page 5-23](#).

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

Starting the Database Server and Initialize Disk Space

UNIX

To bring the database server to on-line mode, enter **oninit**.

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

WIN NT

In Windows NT, the database server runs as a service. Use the **Service** control application to bring the database server to on-line 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 NT is to use the following command where *dbservername* is the database server name:

```
starts dbservername -iy
```



For a description of the types of initialization and associated commands, refer to [Chapter 8, “Managing Database Server Operating Modes.”](#)



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.

Performing Administrative Tasks

After you initialize the database server, you need to perform the following administrative tasks:

- Prepare the operating-system registry or scripts to automatically start and stop the database server.
- Make arrangements for backup management.
- Make sure that users have the correct environment variables.
- Warn the UNIX system administrator about **cron** jobs. ♦

UNIX

WIN NT***Preparing for Automatic Startup on Windows NT***

To start the database server automatically when Windows NT starts, perform the following actions:

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.

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

UNIX***Preparing the UNIX Startup and Shutdown Scripts***

You can modify the UNIX start-up 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.

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 on-line mode.

6. 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.
7. If you are using 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 Immediate-Shutdown and takes the database server off-line.

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 different version.

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

Making Sure That Users Have the Correct Environment Variables

Make sure that every user of an Informix product has the correct environment variables. Each user must set the following environment variables before accessing the database server:

- **INFORMIXDIR**
- **INFORMIXSERVER**
- **PATH**

All users who use database server utilities such as **onstat** must set the **ONCONFIG** environment variable to the name of the ONCONFIG file.

Three techniques are available for setting **INFORMIXDIR**, **INFORMIXSERVER**, **PATH**, and **ONCONFIG**:

- Ask the UNIX administrator to set these environment variables for every user during the login procedure.
- Modify the login procedures for each database server user so that these environment variables are set during login.
- Educate your users to set the environment variables manually every time that they want to work with the database server.

Users might need other environment variables, such as **TERMCAP** and **LC_COLLATE**, to describe their environment fully. You can prepare an environment-configuration file or assign values individually as discussed in [“Other Environment Variables” on page 3-12](#). For a detailed discussion of environment variables, refer to the *Informix Guide to SQL: Reference*.

UNIX

Warning UNIX System Administrator About cron Jobs

The database server creates the **.inf.servicename** and **VP.servername.xx**C files in the **/INFORMIXTMP** directory. Some UNIX systems run **cron** jobs that routinely delete all files from the **/INFORMIXTMP** directory. For information about files that the database server uses, refer to the *Administrator's Reference*.

Setting Up Your Storage Manager and Storage Devices

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. For information on ON-Bar and related configuration parameters, see the *Backup and Restore Guide*.

ON-Bar is packaged with 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 managing backup devices and media, refer to the *Informix Storage Manager Administrator's Guide* or to your third-party storage-manager documentation.

ON-Archive and **ontape** do not require a storage manager. For information on ON-Archive and **ontape**, see the *Archive and Backup Guide*.

Creating Storage Spaces

After the database server is initialized, you can create storage spaces such as dbspaces and sbspaces as desired. For a description of storage spaces, refer to “[Logical Units of Storage](#)” on page 14-19. For a discussion of the allocation and management of storage spaces, refer to [Chapter 15, “Managing Disk Space.”](#)

If you know that your queries require temporary tables for sorts and hash joins, you might want to create temporary dbspaces. For more information on query operations that require temporary tables, refer to “[Temporary Tables](#)” on page 14-38.

Monitoring Configuration Information

One of the tasks of the database server administrator is to keep records of the configuration. Methods of obtaining configuration information are as follows.

Using Command-Line Utilities

Use the following utilities to monitor configuration information.

onstat -c

Execute **onstat -c** to display a copy of the ONCONFIG file. For information about the ONCONFIG file, see the chapter on configuration parameters in the *Administrator's Reference*.

Changes to the ONCONFIG file do not take effect until you shut down and restart the database server, also called *reinitializing shared memory*. If you change a configuration parameter but do not reinitialize shared memory, the effective configuration differs from what the **onstat -c** option displays.

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** on Windows NT.

oncheck -pr

The database server also stores current configuration information in the PAGE_CONFIG reserved page. The reserved page contains a description of the current, effective configuration.

To list the reserved page, execute **oncheck -pr**.

[Figure 3-1 on page 3-23](#) shows sample output.

If you change the configuration parameters from the command line and run **oncheck -pr** before you reinitialize shared memory, **oncheck** discovers that values in the configuration file do not match the current values in the reserved pages and returns a warning message.

```
...
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 3-1
*PAGE_CONFIG
Reserved Page*

UNIX

Using ON-Monitor

Select Configuration from the Status menu. This option creates a copy of the current, effective configuration and stores 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.

If you modify the configuration parameters but have not yet reinitialized shared memory, the effective parameters might be different than the parameters in the ONCONFIG file.

Configuration Parameters

In This Chapter	4-3
Disk-Space Parameters	4-3
Root Dbspace	4-4
Mirror of Root Dbspace	4-5
Other Space-Management Parameters	4-5
Database Server Identification Parameters	4-6
Logging Parameters	4-7
Logical Log	4-7
Physical Log	4-8
Storage-Space and Logical-Log Backups	4-8
Message-Log Parameters	4-9
Shared-Memory Parameters.	4-10
Shared-Memory Size Allocation	4-10
Shared-Memory Space Allocation	4-11
Shared-Memory Buffer Control	4-12
Decision-Support Parameters	4-13
Database Server Process Parameters	4-14
Processor Type	4-14
Processor Affinity	4-15
Time Intervals	4-16

Restore Parameters	4-16
Data-Replication Parameters	4-17
Event-Alarm Parameters	4-17
Dump Parameters	4-18
Specialized Parameters	4-18
Auditing	4-19
Optical Media	4-19
UNIX	4-19

In This Chapter

This chapter provides an overview of the ONCONFIG configuration parameters that the database server uses. 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*.

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 14-22 .
ROOTOFFSET	Specifies an offset. For information about when to set ROOTOFFSET, refer to “Specifying an Offset” on page 15-7 .
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 15-6 .
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 14-45 .

Mirror of Root Dbspace

Mirroring allows fast recovery from a disk failure while the database server remains in on-line 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.
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 15-7 .

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 15-15 .
FILLFACTOR	Specifies how much to fill index pages when indexes are created. For more information, see your Performance Guide .
ONDBSPACEDOWN	Defines how the database server treats a disabled dbspace that is not a critical dbspace.
SBSPACE NAME	Specifies the name of the default sbpace.
SYSSBSPACE NAME	Specifies the name of the sbpace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types.

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	Specifies an alternate name or names for an instance of the database server. For information about using DBSERVERALIASES to create multiple listen endpoints to which clients can connect, see “Listen and Poll Threads for the Client/Server Connection” on page 10-34.
DBSERVERNAME	Specifies the unique name of an instance of the database server. Use the DBSERVERNAME for your INFORMIX-SERVER environment variable and in the sqlhosts information.
SERVERNUM	Specifies a unique integer for the database server instance. The database server uses SERVERNUM to determine the shared-memory segment addresses.



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

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
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 12-21 .
LOGFILES	Specifies the number of logical-log files used to store logical-log records until they are backed up on disk.
LOGSIZE	Specifies the size of each logical-log file.
LOGSMAX	Specifies the maximum (not the actual) number of log files that you expect to have. For more information, see “Adding a Logical-Log File” on page 20-4 .
LTXHWM	Specifies the percentage of the available logical log that can be used before the database server takes moderate action to avoid the undesirable effects of reaching the point of LTXEHW, the long-transaction exclusive-access high-water mark. For more information, see “Logical Log and Long Transactions” on page 19-17 .
LTXEHW	Specifies the point at which the database server takes drastic action.

For more information about these parameters, refer to [“Size and Number of Logical-Log Files” on page 19-7](#).

Physical Log

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 21, “Physical Logging.”](#)

Storage-Space and Logical-Log Backups

To create storage-space and logical-log backups of database server data, you can use the following tools. To verify storage-space backups, use ON-Bar.

Tool	Reference
ON-Bar	<i>Backup and Restore Guide</i>
ON-Archive (UNIX)	Archive and Backup Guide
ontape utility	<i>Administrator’s Reference</i>

If you use the **ontape** utility, use the following parameters to describe your tape devices.

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 2-15 .
MSGPATH	Specifies the pathname of the database server message-log file. For more information, refer to “Message Log” on page 2-13 .

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 12, “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 NT.

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 12-18 .
CKPTINTVL	Specifies the maximum time interval allowed to elapse before a checkpoint.
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 Performance Guide .
DD_HASHSIZE	Specifies the number of hash buckets in the data-dictionary cache. For more information about setting DD_HASHSIZE, refer to your Performance Guide .
LBU_PRESERVE	Specifies the logs-full high-water mark. This feature preserves log space for administrative tasks. For a discussion of the logs-full high-water mark, refer to “Setting High-Water Marks” on page 19-20 .
LOCKS	Specifies the initial number of locks available to database server user processes during transaction processing.
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 Performance Guide .
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 Performance Guide .
RESIDENT	Specifies whether shared-memory residency is enforced.

(1 of 2)

Configuration Parameter	Description
STACKSIZE	Specifies the stack size for database server user threads. For a discussion of the use of stacks, refer to “ Stacks ” on page 12-31 .
STMT_CACHE	Specifies whether users can hold a parsed and optimized SQL statement in the SQL statement cache in memory. For more information about the SQL statement cache, refer to your Performance Guide .
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.

(2 of 2)

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	Describe 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 12-36 .
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 12-45 .
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 12-41 .

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.

Processor Type

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
- NUMAIOVPS
- NUMCPUVPS
- SINGLE_CPU_VP

For guidelines on setting these parameters, refer to [“Setting Virtual-Processor Configuration Parameters with a Text Editor”](#) on page 11-4.

Configuration Parameter	Description
MULTIPROCESSOR	Specifies the appropriate type of locking.
NETTYPE	Provides tuning options for each communications protocol.
NOAGE	Specifies whether priority aging should be in effect.
NUMAIOVPS	Specifies the number of virtual processors for AIO (asynchronous I/O) class threads.

(1 of 2)

Configuration Parameter	Description
NUMCPUVPS	Specifies the number of virtual processors for CPU class threads.
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. You can use the first six parameters listed in the previous table or the VPCLASS parameter. Informix recommends that you use the VPCLASS parameter.

(2 of 2)

Processor Affinity

Some multiprocessor computers support *processor affinity*, which allows you to bind CPU virtual processors to CPUs. Use the following parameters to establish processor affinity.

Configuration Parameter	Description
AFF_NPROCS	On multiprocessor computers that support <i>processor affinity</i> , specifies the number of CPUs to which the database server can bind CPU virtual processors.
AFF_SPROC	Specifies the CPU on which the database server starts binding CPU virtual processors to CPUs.

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 off-line and on-line logical recovery.

Configuration Parameter	Description
OFF_RECOVERY_THREADS	Specifies the number of recovery threads used during a cold restore.
ON_RECOVERY_THREADS	Specifies the number of recovery records used during fast recovery and a warm restore.

Data-Replication Parameters

Use the high-availability data-replication parameters to control the behavior of a pair of high-availability data-replication servers.

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, simple large object, or smart large object; chunk or dbspace taken off-line; internal subsystem failure; initialization failure; and detection of long transaction.

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.

The ALARMPROGRAM configuration parameter specifies the location of the executable file. You can set the ALARMPROGRAM parameter to back up logs automatically as they fill. For more information, refer to the *Backup and Restore Guide*.

Dump Parameters

Use the following parameters to control the types and location of core dumps that are performed if the database server fails.

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

Use the auditing parameters only when you use the database server auditing features. The [Trusted Facility Manual](#) 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

Use the following configuration parameters when you use the Optical Subsystem. For more information about these parameters, refer to the [Guide to the Optical Subsystem](#).

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.

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

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

UNIX

Client/Server Communications

In This Chapter	5-3
Client/Server Architecture	5-3
Network Protocol	5-4
Network Programming Interface.	5-5
Windows NT Network Domain	5-5
Database Server Connection	5-6
Multiplexed Connection.	5-7
Connections That the Database Server Supports.	5-9
Local Connections.	5-10
Shared-Memory Connections	5-11
Stream-Pipe Connections	5-12
Named-Pipe Connections	5-12
Local-Loopback Connections.	5-13
Communication Support Services.	5-13
Informix Password Communication Support Module	5-14
Connectivity Files	5-14
Network-Configuration Files	5-15
TCP/IP Connectivity Files	5-15
Multiple TCP/IP Ports	5-17
IPX/SPX Connectivity Files	5-18
Network-Security Files	5-18
The hosts.equiv File	5-18
The netrc Information	5-19
CSM Configuration File	5-21
Format of the CSM Configuration File	5-21
The conccsm.cfg Entry for Password Encryption	5-22
The sqlhosts File and the SQLHOSTS Registry Key	5-23
The sqlhosts File	5-23
The SQLHOSTS Registry Key	5-24

The sqlhosts Information	5-27
Connectivity Information	5-28
Database Server Name	5-28
The Connection Type Field	5-29
Host Name Field	5-31
Service Name Field	5-32
Options Field	5-34
Group Information.	5-41
Database Server Group	5-41
Group Keyword in the Connection-Type Field	5-41
Alternatives for TCP/IP Connections	5-42
IP Addresses for TCP/IP Connections	5-42
Wildcard Addressing for TCP/IP Connections.	5-43
Port Numbers for TCP/IP Connections	5-46
ONCONFIG Parameters for Connectivity	5-46
DBSERVERNAME Configuration Parameter.	5-47
DBSERVERALIASES Configuration Parameter	5-47
NETTYPE Configuration Parameter.	5-48
Environment Variables for Network Connections	5-49
Examples of Client/Server Configurations.	5-49
Using a Shared-Memory Connection	5-50
Using a Local-Loopback Connection	5-51
Using a Network Connection	5-52
Using Multiple Connection Types	5-53
Accessing Multiple Database Servers	5-55

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

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 NT is WINSOC (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 5-6](#).

To specify which protocol the database server uses, set the **nettype** field in the **sqlhosts** file on UNIX. On Windows NT, set the PROTOCOL field in the SQLHOSTS registry key. For more information, see [“The sqlhosts File and the SQLHOSTS Registry Key” on page 5-22](#).

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 application-programming interface. 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 5-52](#).

WIN NT

Windows NT Network Domain

Windows NT network technology enables you to create network *domains*. A domain is a group of connected Windows NT 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 NT database server, a user on another Windows NT 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 trusting domain.

A user who attempts to log in to a Windows NT computer that is a member of a domain can either do so 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 NT domain in which the database server runs. This rule also applies to trusted domains.

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



Important: The Informix trusted client mechanism is unrelated to the trust relationship that you can establish between Windows NT domains. Therefore, even if a client connects from a trusted Windows NT domain, Informix requires the user to 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 5-13](#) and [“Network-Security Files” on page 5-18](#).

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 '@myserver'
```

For more information on the `CONNECT` and `DATABASE` statements, see the [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 5-22](#).

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 NT	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. To check which interface/protocol combinations the database server supports for your operating system, check the machine notes. The section “Machine Specific Notes” describes interface/protocol combinations that are available on your platform, similar to the following example:

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 NT 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 5-14](#)
- [“The sqlhosts Information” on page 5-27](#)
- [“ONCONFIG Parameters for Connectivity” on page 5-46](#)
- [“Environment Variables for Network Connections” on page 5-49](#)

Local Connections

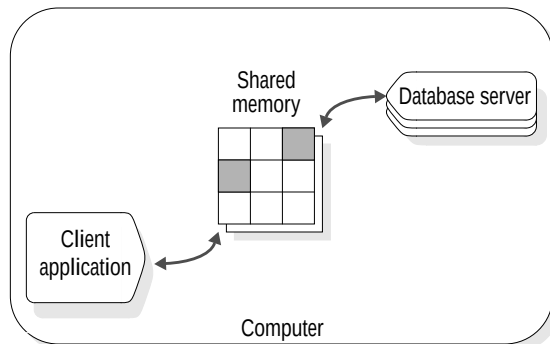
A *local connection* is a connection between a client and the database server on the same computer. The following sections describe these types of local connections.

UNIX

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 5-1](#) illustrates a shared-memory connection.

Figure 5-1*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 5-50](#).

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

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 12-33](#). For additional information, you can also refer to [“How a Client Attaches to the Communications Portion” on page 12-11](#).

UNIX

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” on page 5-30](#) and [“Shared-Memory and Stream-Pipe Communication” on page 5-32](#).

Advantages of Stream-Pipe Connections

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.

Disadvantages of Stream-Pipe Connections

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.

WIN NT

Named-Pipe Connections

Named pipes are application programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows NT. 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 5-51](#).

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 5-18](#). 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).

Informix Password Communication Support Module

The Informix Password Communication Support Module (PSWDCSM) provides encryption to protect a password when it must be sent between the client and the database server for authentication. PSWDCSM is available on all platforms.

To use password 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 that you are using. For information on the **concsm.cfg** file, see [“CSM Configuration File” on page 5-21](#). For information on specifying the CSM in the **sqlhosts** file or registry, see [“Communication Support Module Option” on page 5-36](#).

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 5-8 on page 5-42](#) 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
Informix products use this name to determine the port number and protocol for making client/server connections. The service name can have up to 256 characters.
- Port number and protocol
The port number is the computer port, and the protocol for TCP/IP is `tcp`.
- 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 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 5-6 on page 5-33](#) 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 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
```



WIN NT

TCP/IP Connectivity Files on Windows NT

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) assigns IP addresses dynamically 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
```

All ports in use for a single IP address must be unique. Separate ethernet cards can utilize the same or different port numbers. You might want to use the same port number on each ethernet card because you are 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 NT, 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

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 NT, 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.

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.

WIN NT

On Windows NT, a home directory is not automatically assigned when the Windows NT 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 NT maintains the **netrc** information in the registry keys. Use the Host Information tab of **setnet32** to edit the **netrc** information.

WIN NT

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.

On Windows NT, a home directory is not automatically assigned when the Windows NT 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"

CSM Configuration File

The **concsbm.cfg** file describes the communication support module (CSM) and is required only if you use a 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.

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 [Informix Guide to SQL: Reference](#).

Format of the CSM Configuration File

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 CSM is normally installed in **\$INFORMIXDIR/lib/csm**, and the database server uses the CSM to communicate with the clients. The following restrictions apply to the CSM pathnames:

- Characters '=', '" and ',' are not allowed to be part of the pathname.
- 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 PWDCSM.

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.

The conccsm.cfg Entry for Password Encryption

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

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

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

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

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


UNIX

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 5-27](#).

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. You cannot 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 5-2](#) shows a sample **sqlhosts** file.

Figure 5-2
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	

(1 of 2)

WIN NT

dbservername	nettype	hostname	servicename	options
asia	group	—	—	e=asia.3
asia.1	ontlircp	node6	svc8	g=asia
asia.2	onsocrcp	node0	svc1	g=asia

(2 of 2)

The SQLHOSTS Registry Key

When you install the database server, the **setup** program creates the following key in the Windows NT 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 NT 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 NT 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 NT documentation.

Preparing the SQLHOSTS Registry Key

To manage the SQLHOSTS registry key, use one of the following tools:

- Client Connectivity tool
- **setnet32**
- **regedt32**

For more information about the Client Connectivity tool and **setnet32**, refer to the documentation notes described in [“Documentation Notes, Release Notes, Machine Notes”](#) on page 14 of the Introduction.



Tip: Informix recommends that you use the Client Connectivity tool or **setnet32** to manage the SQLHOSTS connectivity information.

To add connectivity information with regedt32

1. Run **regedt32**.
2. In the Registry Editor window, select the window for the HKEY_LOCAL_MACHINE subtree.
3. Click the folder icons to select the **\SOFTWARE\INFORMIX\SQLHOSTS** branch.
4. With the SQLHOSTS key selected, choose **Edit→Add Key**.
5. In the **Add Key** dialog box, enter the name of the database server in the **Key Name** box. Leave the **Class** box blank. Click **OK**.
6. Select the new key that you just made (the key with the database server name). Choose **Edit→Add Value**.

7. In the **Add Value** dialog box, enter one of the fields of the **sqlhosts** information (HOST, OPTIONS, and so on) in the Value Name box. Do not change the Data Type box. Click **OK**.
8. In the **String Editor** box, type the new value for the field that you selected and click **OK**.
9. Repeat steps 7 and 8 for each field of the **sqlhosts** information.
10. Exit the Registry Editor.

Figure 5-3 illustrates the location and contents of the SQLHOSTS registry key for the database server **payroll**.

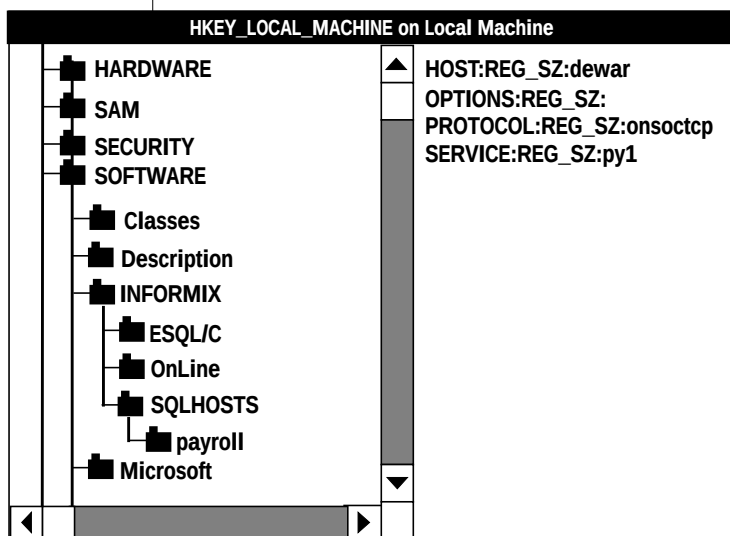


Figure 5-3
*sqlhosts Information in
the Windows NT Registry*

The sqlhosts Information

The **sqlhosts** information in the **sqlhosts** file on UNIX or the SQLHOSTS registry key on Windows NT 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 NT, 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 NT 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 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 *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 5-46](#).

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.

UNIX

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** on UNIX or **PROTOCOL** on Windows NT) 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 5-4](#) shows.

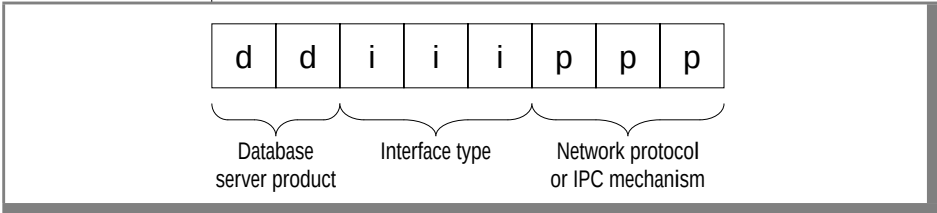


Figure 5-4
Format of the Connection-Type Field

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

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)	Informix Enterprise Gateway with DRDA For information about DRDA, refer to the <i>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 5-5](#).

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

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

Protocol Subfield	Type of Protocol
shm	Shared-memory communication
spx	IPX/SPX network protocol
str	Stream-pipe communication
nmp	Named-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 5-5](#) summarizes the possible connection-type values for database server connections.

Figure 5-5
Summary of nettype Values

nettype value (UNIX)	PROTOCOL value (Windows NT)	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
ontlispX		TLI with IPX/SPX protocol	Network

For information on the connection types for your platform, see [“Connections That the Database Server Supports” on page 5-9](#).

Host Name Field

The host name field (**hostname** on UNIX or **HOST** on Windows NT) 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 5-15](#).

UNIX

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 5-42](#).

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 5-6 shows the relationship between the **sqlhosts** file or registry and the **hosts** file, as well as the relationship of **sqlhosts** to the services file.

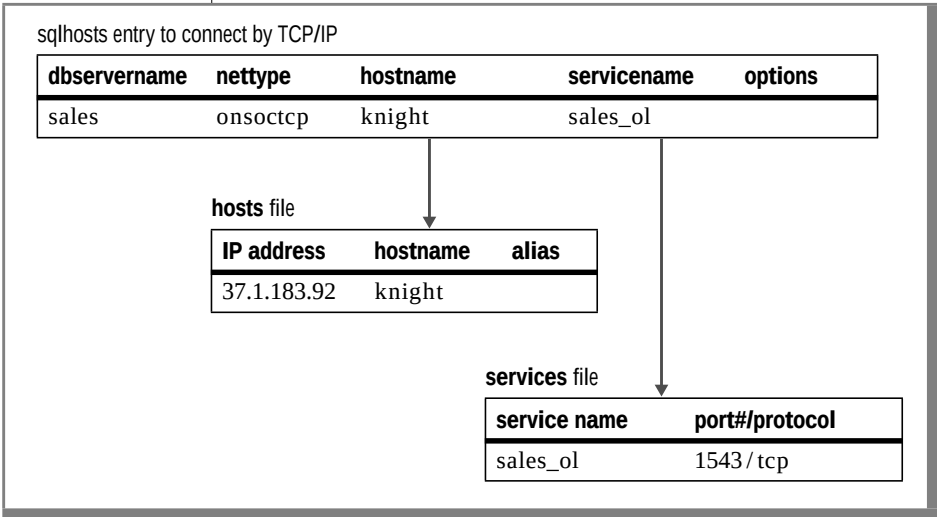


Figure 5-6
*Relationship of
sqlhosts File or
Registry to hosts
and services Files*

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 5-46.

WIN NT

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. Informix recommends that you use the **dbservername** as the **servicename** for stream-pipe connections.

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 5-35
End of group	e	page 5-36
Group	g	page 5-37
Identifier	i	page 5-39
Keep-alive	k	page 5-39
Security	s (database server) r (client)	page 5-40
Communication support module	csm	page 5-36

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: Informix recommends 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.

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:

```
cs m=( csmname, csm-connection-options )
```

The value of *csmname* must match a *csmname* entry in the **conccsm.cfg** file. The *connection-options* parameter overrides the default *csm-connection* options specified in the **conccsm.cfg** file. For information on the **conccsm.cfg** file entry, refer to [“CSM Configuration File” on page 5-21](#).

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

```
cs m=( PSWDCSM )
```

For more information on the CSM, refer to [“Database Server Connection” on page 5-6](#). For more information on the **conccsm.cfg** file, refer to [“CSM Configuration File” on page 5-21](#).

End of Group Option

Use this option to specify the ending database server name of a database server group. You can use this option only in a group entry. 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.

Group Option

When you define database server groups in the **sqlhosts** file key, you can use multiple related entries as one logical entity to establish or change client/server connections. Enterprise Replication uses groups to declare database servers for replication. Use the following steps to create database server groups.

To name 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 only options available for a database server group entry are as follows:

- e = end of group
- i = identifier option



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

Figure 5-7 shows **sqlhosts** entries that define database server groups.

Figure 5-7
Database Server Groups in sqlhosts File or Registry

dbservername	nettype	hostname	servicename	options
asia	group	—	—	e=asia.3
asia.1	ontlircp	node6	svc8	g=asia
asia.2	onsoctcp	node0	svc1	g=asia
usa.2	ontlirpx	node9	sv2	
asia.3	onsoctcp	node1	svc9	g=asia

(1 of 2)

dbservername	nettype	hostname	servicename	options
peru	group	—	—	
peru.1	ontlitcp	node4	svc4	
peru.2	ontlitcp	node5	svc5	g=peru
peru.3	ontlitcp	node7	svc6	
usa.1	onsoctcp	37.1.183.92	sales_ol	k=1, s=0
asia	group	—	—	e=asia.3
asia.1	ontlitcp	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	—	—	

(2 of 2)

The example in [Figure 5-7](#) 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.

You can use the name of a database server group instead of the database server name in the following environment variables:

- **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.

In addition, the group name can also be in the SQL CONNECT command.

For more information on the use of the group option, refer to the [Guide to Informix Enterprise Replication](#).

Identifier Option

The identifier option 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 [Guide to Informix Enterprise Replication](#).

The keep-alive option is a network option that TCP/IP uses. It does not affect other types of connections.

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, the keep-alive option enables the network service to check 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 connection is considered broken, and all resources related to the connection are released.

When the keep-alive option is enabled, the network service spends additional resources to check the connection.

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.

If you do not include the keep-alive option in the **options** field, the keep-alive feature is enabled by default. You can set this option on the database server side only, the client side only, or on both sides. For most cases, Informix recommends that you enable the keep-alive option.

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
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
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 servers 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 5-37](#).

Group Keyword in the Connection-Type Field

The **group** keyword begins a list of coservers that are part of the database server. When a client connects to the database server using the database server name, the database server selects a coserver on which to establish the client connection.

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 Internet IP address in the **hostname** field instead of the host name or alias found in the **hosts** file. The IP address is always composed of four sets of one to three integers, separated by periods. [Figure 5-8](#) shows sample IP addresses and host from a **hosts** file.

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

Figure 5-8
A Sample hosts File

Using the IP address for **knight** from [Figure 5-8](#), the following two **sqlhosts** entries are equivalent:

```
sales    ontlitcp    12.34.56.789    sales_01
sales    ontlitcp    knight          sales_01
```

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 NT 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 ♦

UNIX

WIN NT

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).

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 5-9 on page 5-44](#), 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

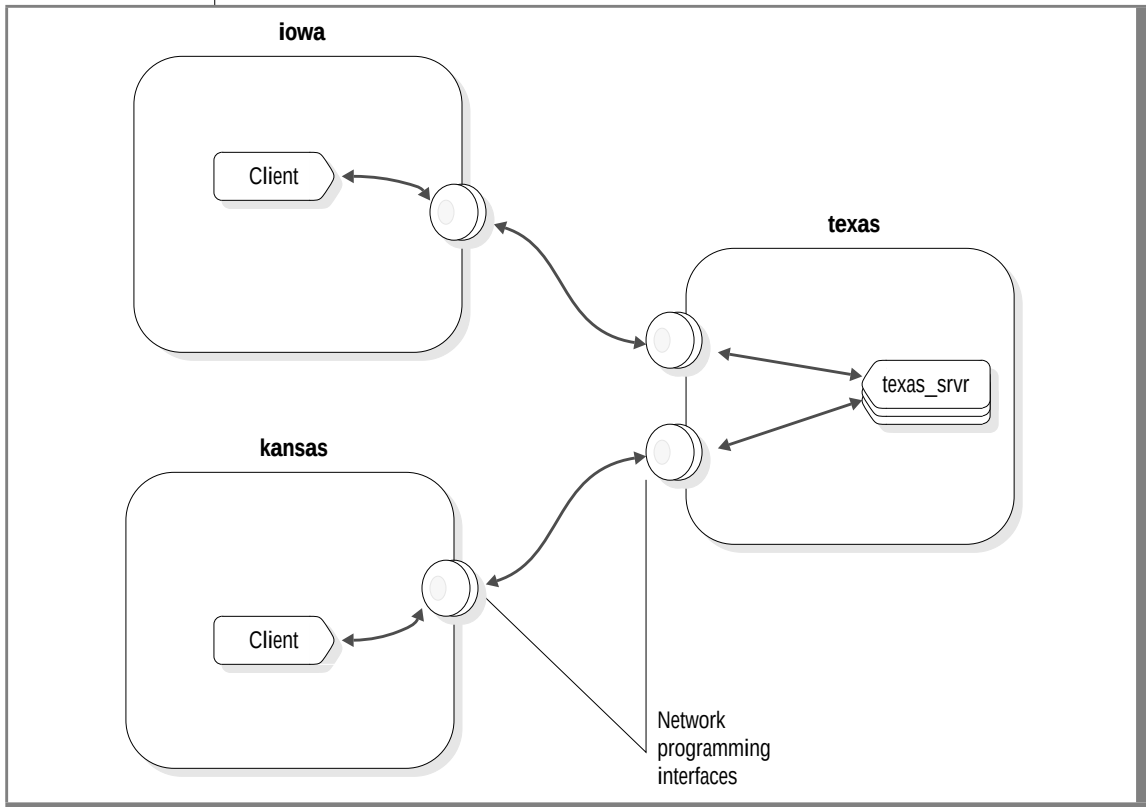
You can use the wildcard (*) alone or as a prefix for a host name or IP address, as shown in [Figure 5-10 on page 5-45](#).

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 5-9 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 5-9
Using Multiple Network-Interface Cards



The connectivity information for the **texas_srvr** database server can be any of the entries in Figure 5-10.

Figure 5-10
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, Informix recommends 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 an 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.

Using the port number for the **sales** database server from the **services** file in [Figure 5-6](#), an entry in **sqlhosts** could also be written as in the following example.

servername	nettype	hostname	servicename
sales	ontlitcp	knight	1536

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

In Windows NT, 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. [Figure 5-11](#) shows entries in an ONCONFIG configuration file that assign three *dbservernames* to the same database server instance.

Figure 5-11

Example of DBSERVERNAME and DBSERVERALIASES Parameters

DBSERVERNAME	sockets_srvr
DBSERVERALIASES	ipx_srvr,shm_srvr

WIN NT

The **sqlhosts** entries associated with the dbservernames from [Figure 5-11](#) could include those shown in [Figure 5-12](#). 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 5-12

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 5-12](#), 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, ipcnmp, soctcp, tlitcp, and tlisp) can have a separate NETTYPE entry in the configuration file.

Although the NETTYPE parameter is not a required parameter, Informix recommends 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 10-32](#). For additional information, regarding the impact of 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 12-11.

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 [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.

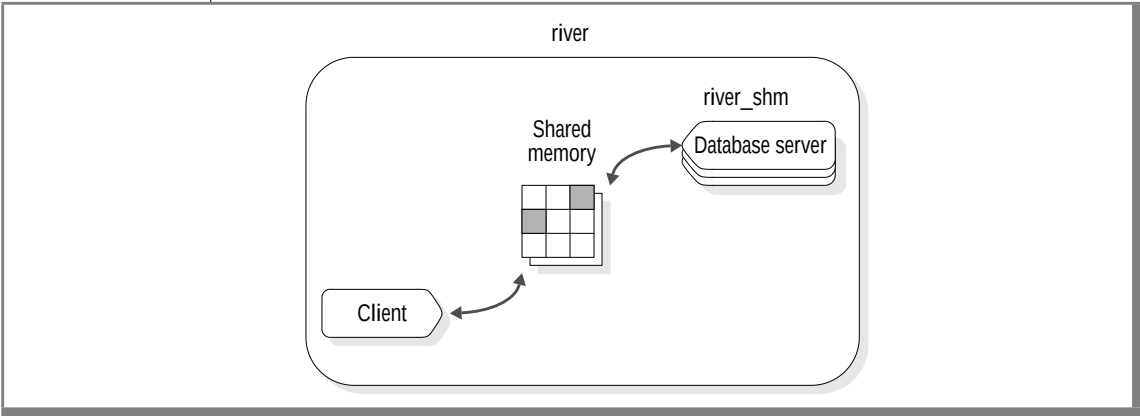


UNIX

Using a Shared-Memory Connection

Figure 5-13 shows a shared-memory connection on the computer named **river**.

Figure 5-13
A Shared-Memory Connection



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

```
DBSERVERNAME river_shm
```

A correct entry for the **sqlhosts** file or SQLHOSTS registry key is as follows.

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 12-11.

Using a Local-Loopback Connection

Figure 5-14 shows a local-loopback connection. The name of the host computer is **river**.

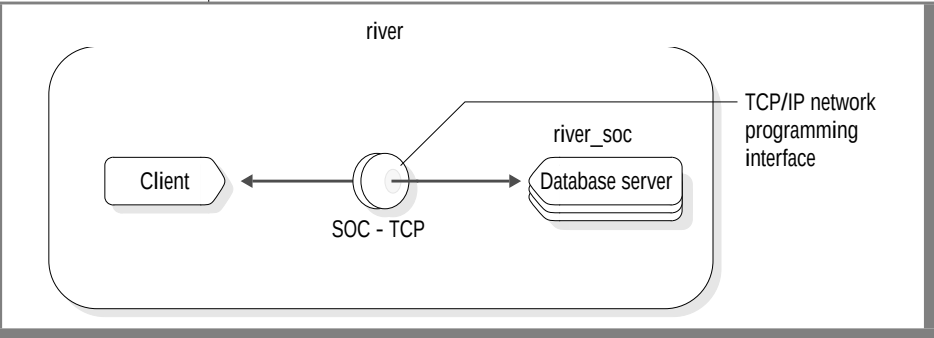


Figure 5-14
Local-Loopback
Connection

The network connection in Figure 5-14 uses sockets and TCP / IP, so the correct entry for the **sqlhosts** file or SQLHOSTS registry key is as follows.

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 **ontlntcp** instead of **onsoctcp**.

The ONCONFIG file includes the following line:

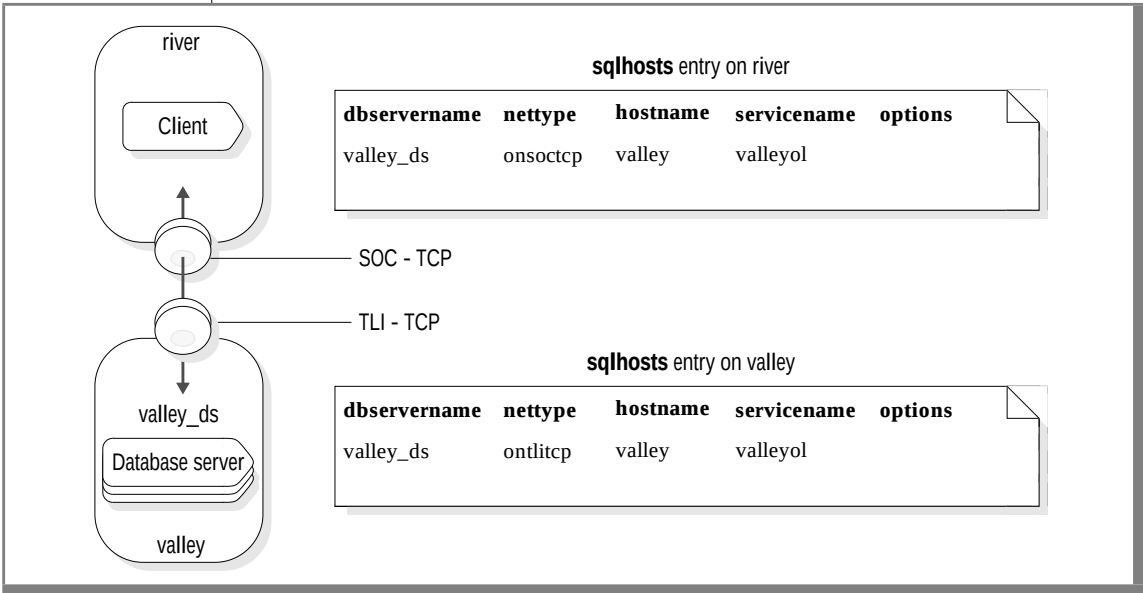
```
DBSERVERNAME river_soc
```

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 5-15 shows a configuration in which the client application resides on host **river** and the database server resides on host **valley**.

Figure 5-15
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. If the configuration in [Figure 5-15 on page 5-52](#) uses IPX/SPX instead of TCP/IP, the entry in the **sqlhosts** file on both computers is as follows.

dbservername	nettype	hostname	servicename
valley_us	ontlspix	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 5-16 on page 5-54](#) 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:

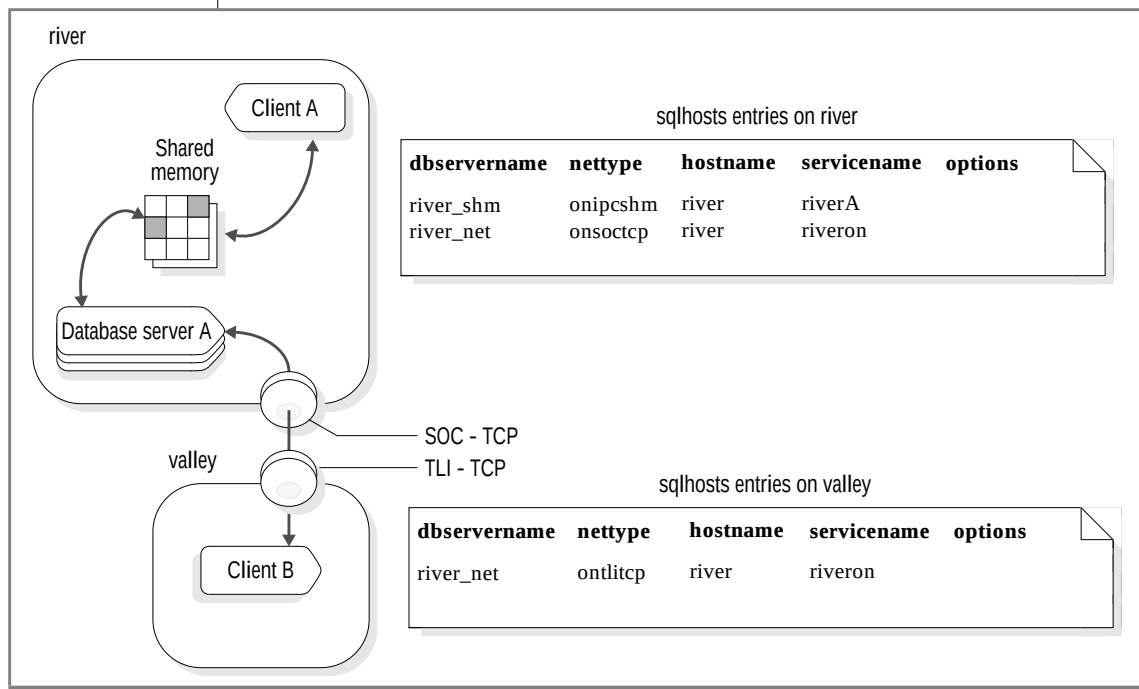
- Put DBSERVERNAME and DBSERVERALIASES entries in the ONCONFIG configuration file.
- Put an entry in the **sqlhosts** file or SQLHOSTS registry key for each database server/connection type pair.

For the configuration in [Figure 5-16 on page 5-54](#), the database server has two dbservernames: **river_net** and **river_shm**. The ONCONFIG configuration file includes the following entries:

```
DBSERVERNAME      river_net
DBSERVERALIASES   river_shm
```

Figure 5-16

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:

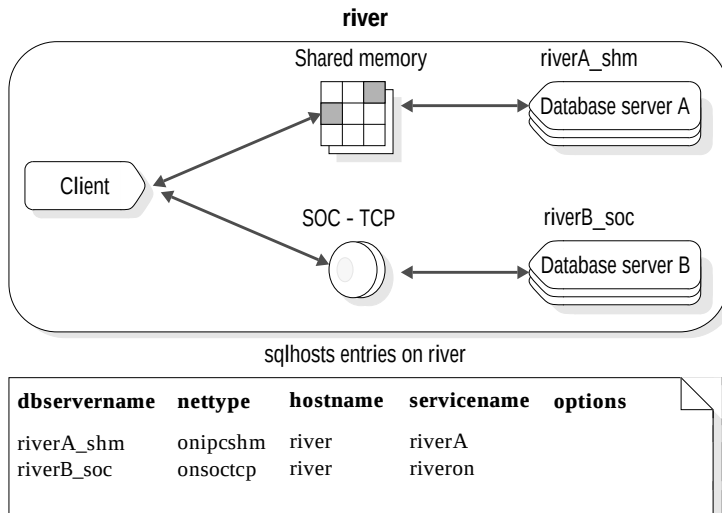
```
CONNECT TO '@river_net'
```

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 5-17 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 Chapter 6, “Multiple Residency.”)

Figure 5-17
Multiple Database Servers on UNIX



For the configuration in Figure 5-17, 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:

```
DBSERVERNAME      riverA_shm
```

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

```
DBSERVERNAME      riverB_soc
```

Multiple Residency

In This Chapter	6-3
Benefits of Multiple Residency	6-3
How Multiple Residency Works	6-4
The Role of the ONCONFIG Environment Variable	6-5
The Role of the SERVERNUM Configuration Parameter.	6-5

In This Chapter

You can use more than one database server in the following two ways:

- Run multiple instances of the database server on a single host computer
- Access several database servers over a network

When multiple database servers and their associated shared memory and disk structures coexist on a single computer, it is called *multiple residency*. This chapter covers the concept of multiple residency for the database server.

Benefits of Multiple Residency

Creating independent database server environments on the same computer allows you to perform the following actions:

- Separate production and development environments
- Isolate sensitive databases
- Test distributed data transactions on a single computer

When you use multiple residency, each database server has its own configuration file. Thus, you can create a configuration file for each database server that meets its special requirements for backups, shared-memory use, and tuning priorities.

You can separate production and development environments to protect the production system from the unpredictable nature of the development environment. You might also find it useful to isolate applications or databases that are critically important, either to increase security or to accommodate more frequent backups than most databases require.

If you are developing an application for use on a network, you can use local loopback to perform your distributed-data simulation and testing on a single computer. (See [“Using a Local-Loopback Connection” on page 5-51.](#)) Later, when a network is ready, you can use the application without changes to application source code.

However, running multiple database servers on the same computer is not as efficient as running one database server. You need to balance the advantages of separate database servers against the extra performance cost.

How Multiple Residency Works

Multiple residency is possible because the operating system can maintain separate areas in shared memory and on disk for each instance of the database server. Each instance of the database server passes a value to the operating system. This value, which is a function of the SERVERNUM configuration parameter, specifies the shared-memory address to which the database server process should attach. You must also specify a unique database server name and unique storage locations for each instance of the database server.

The Role of the ONCONFIG Environment Variable

The parameters in an ONCONFIG configuration file describe each instance of the database server. The **ONCONFIG** environment variable specifies the name of the current ONCONFIG configuration file. The following configuration parameters should have unique values for each database server:

- DBSERVERALIASES (if used)
- DBSERVERNAME
- MIRRORPATH
- MSGPATH
- ROOTPATH
- SERVERNUM

For instruction on setting these parameters, see [Chapter 7, “Using Multiple Residency.”](#)

The Role of the SERVERNUM Configuration Parameter

To maintain separation among the instances of database servers, you maintain multiple configuration files, each with a unique SERVERNUM value. When you initialize the database server, it reads the **ONCONFIG** environment variable for the name of its configuration file. Next, the database server reads its configuration file to obtain the value of its SERVERNUM configuration parameter. The database server then uses the SERVERNUM value to calculate the required shared-memory address.

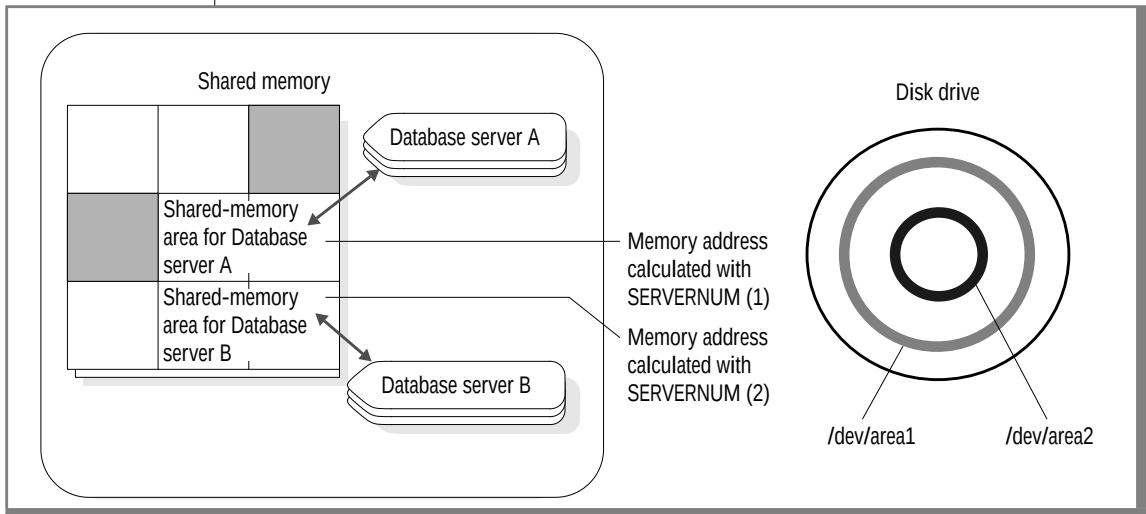
For example, the ONCONFIG files for two database servers on a UNIX platform might include these parameters.

ONCONFIG file: onconfig.one	ONCONFIG file: onconfig.two
...	...
DBSERVERNAME dbsrvr_one	DBSERVERNAME dbsrvr_two
SERVERNUM 1	SERVERNUM 2
ROOTPATH /dev/area1	ROOTPATH /dev/area2
...	...

Figure 6-1 uses the configuration files shown in the preceding table to provide an example of multiple residency. Each database server has its own name, its own section of shared memory, and its own storage area on disk.

Figure 6-1

Separate Memory and Storage in Multiple Residency



Using Multiple Residency

Planning for Multiple Residency	7-3
Creating a New Database Server on Windows NT	7-5
Adding an Instance of the Database Server	7-6
Deleting an Instance of the Database Server	7-7
Creating a New Database Server on UNIX.	7-7
Prepare a Configuration File	7-8
Set the ONCONFIG Environment Variable	7-8
Edit the New Configuration File	7-8
Add Connection Information	7-10
Update the sqlhosts File	7-10
Initialize Disk Space	7-10
Prepare Dbpace and Logical-Log Backup Environment.	7-11
Update the Operating-System Boot File	7-12
Check INFORMIXSERVER Environment Variables for Users	7-12

In This Chapter

This chapter describes how to use multiple database servers on the same computer. It includes the following topics:

- Planning for multiple residency
- Creating a database server instance on Windows NT
- Creating a database server instance on UNIX

Before you set up multiple residency, you should install one database server as described in [Chapter 3, “Installing and Configuring the Database Server.”](#)

Planning for Multiple Residency

When you plan for multiple residency on a computer, consider the following factors:

■ Storage space

Each database server must have its own unique storage space. You cannot use the same disk space for more than one instance of a database server. When you prepare an additional database server, you need to repeat some of the planning that you did to install the first database server. For example, you need to consider these questions:

- ❑ Will you use buffered or unbuffered files? Will the unbuffered files share a disk partition with another application? (For more information on buffered and unbuffered files, see [“Direct Disk Access” on page 1-11.](#))
- ❑ Will you use mirroring? Where will the mirrors reside?
- ❑ Where will the message log reside?
- ❑ Can you dedicate a tape drive to this database server for its logical logs?
- ❑ What kind of backups will you perform?

■ Memory

Each database server has its own memory. Can your computer handle the memory usage that an additional database server requires?

The following sections present the steps to create an additional database server, or multiple residency, on your computer.



Important: Do not try to install another copy of the database server binary files. All instances of the same version of the database server on one computer share the same executable files.

WIN NT

Creating a New Database Server on Windows NT

On Windows NT, the database server provides a tool that automates the task of creating a new database server instance.

Run **instmgr.exe** from the command line to create an additional instance of the database server on that computer automatically. The **instmgr.exe** program uses a graphical interface to prompt you for the information that it needs.

You must be a member of the **Informix-Admin** group to execute **instmgr.exe**.

Important: *On Windows NT, if you install multiple instances of the database server on one computer, all instances must be Version 7.3 or later. Earlier versions of the database server on Windows NT do not support multiple residency.*

You can execute **instmgr.exe** in one of the following ways:

- Choose **Start→Programs→Informix Administration Tools group**.
- Locate **instmgr.exe** with Windows NT Explorer and click it.
The **instmgr.exe** file is in the %INFORMIXDIR%\bin directory.
- Enter **instmgr** on an MS-DOS command line.

The graphical interface for the **instmgr.exe** program displays the names of the database servers that are already installed and gives you the option to create an additional instance of the database server or delete an instance that already exists.

For information about installing a database server initially and adding an instance of the database server, refer to your [Installation Guide](#).



Adding an Instance of the Database Server

When you add an instance of the database server, you have the option to select one of the following types of installations:

- Typical
- Compact
- Custom

The Typical option is the default selection.

When you click the **Create** button, the **instmgr.exe** program prompts you to enter values for **Server Number**, **Server Name**, **Service Name**, **Port Number**, **Primary Storage Location**, **Password** of the system administrator, and **Shared Server** computer name.

Entry	Purpose
Server Number	Sets the value for the SERVERNUM configuration parameter.
Server Name	Sets the value for DBSERVERNAME configuration parameter.
Service Name	Specifies the servicename entry for the database server in the sqlhosts registry.
Port Number	Specifies the port entry for the database server in the sqlhosts registry.
Primary Storage Location	Specifies the location of the ROOTPATH configuration parameter.
Password of the system administrator	Ensures that you have administrator privileges to add the new database server instance.
Shared Server computer name	Specifies where to update the sqlhosts registry-information for the new database server instance.

After you have entered the required information, the **instmgr.exe** program proceeds to install services, record environment variables, update the registry, and create an ONCONFIG file for the new database server instance.

UNIX

Deleting an Instance of the Database Server

To delete an instance of the database server, select a database server instance from the list and click the **Delete** button.

The **instmgr.exe** program first checks to see if on-line services for the database server are running and, if so, stops them. Then it removes the files and directories, icons, services, registry keys, dbspaces, and configuration information for the database server.

Creating a New Database Server on UNIX

Perform the following steps manually to create an additional instance of the database server.

To create multiple residency of a database server

1. Prepare a new **ONCONFIG** configuration file.
2. Set the **ONCONFIG** environment variable to the new filename.
3. Edit the new **ONCONFIG** configuration file.
4. If needed, add a servicename to the **services** file or connection information to the NetWare server.
5. Update the **sqlhosts** file to include the dbservername(s) of the new database server.
6. Initialize disk space for the new database server.

To ensure proper backups, startup, and client connections

1. Prepare dbspace and backup schedules.
2. Modify the operating-system boot file.
3. Check the **INFORMIXSERVER** environment variables for users.

The following sections describe each of these steps.

Prepare a Configuration File

Each instance of the database server must have its own ONCONFIG configuration file. Make a copy of an ONCONFIG file that has the basic characteristics that you want for your new database server. Give the new file a name that you can easily associate with its function. For example, you might select the filename **onconfig.acct** to indicate the configuration file for a production system that contains accounting information.

Set the ONCONFIG Environment Variable

Set the **ONCONFIG** environment variable to the filename of the new ONCONFIG file. Specify only the filename, not the complete path.

Edit the New Configuration File

To edit the new ONCONFIG file, use a text editor or ON-Monitor.



Warning: If you use ON-Monitor, you must set the **ONCONFIG** environment variable to the name of the new configuration file, and you must change **SERVERNUM** in the file (with a text editor) before you enter ON-Monitor. If not, you will edit the values of the wrong configuration file.

In the new configuration file, you must change the following configuration parameters:

- **SERVERNUM**

The **SERVERNUM** parameter specifies an integer (between 0 and 255) associated with a database server configuration. Each instance of a database server on the same host computer must have a unique **SERVERNUM** value. For more information, refer to [“The Role of the SERVERNUM Configuration Parameter” on page 6-5](#).

- **DBSERVERNAME**

The **DBSERVERNAME** parameter specifies the dbservername of a database server. Informix suggests that you choose a name that provides information about the database server, such as **ondev37** or **hostnamedev37**. For more information, refer to [“DBSERVERNAME Configuration Parameter” on page 5-47](#).

- MSGPATH

The MSGPATH parameter specifies the pathname of the message file for a database server. You should specify a unique pathname for the message file because database server messages do not include the dbservername. If multiple database servers use the same MSGPATH, you cannot identify the messages from separate database server instances. For example, if you name the database server **ondev37**, you might specify **/usr/informix/dev37.log** as the message log for this instance of the database server.

- ROOTPATH and ROOTOFFSET

The ROOTPATH and ROOTOFFSET parameters together specify the location of the root dbspace for a database server. The root dbspace location must be unique for every database server configuration.

If you put several root dbspaces in the same partition, you can use the same value for ROOTPATH. However, in that case, you must set ROOTOFFSET so that the combined values of ROOTSIZE and ROOTOFFSET define a unique portion of the partition. For more information about ROOTPATH and ROOTOFFSET, refer to the chapter on configuration parameters in the *Administrator's Reference*.



Tip: You do not need to change ROOTNAME. Even if both database servers have the name **rootdbs** for their root dbspace, the dbspaces are unique because ROOTPATH specifies a unique location.

You might also need to set the MIRRORPATH and MIRROROFFSET parameters. If the root dbspace is mirrored, the location of the root dbspace mirror must be unique. For information about setting MIRRORPATH, refer to [“Steps Required for Mirroring Data” on page 25-3](#).

Add Connection Information

If you use the TCP/IP communication protocol, you might need to add an entry to the **services** file for the new database server instance. If you use the IPX/SPX communication protocol, you might need to modify the connection information for the NetWare server.

Update the sqlhosts File

The **sqlhosts** file must have an entry for each database server. If Informix products on other computers access this instance of the database server, the administrators on those computers must update their **sqlhosts** files. [Chapter 5, “Client/Server Communications,”](#) discusses the preparation of the **sqlhosts** file.

If you plan to use TCP/IP network connections with an instance of a database server, the system network administrator must update the **hosts** and **services** files. If you use an IPX/SPX network, the NetWare administrator must update the NetWare file-server information. For information about these files, refer to [“Network-Configuration Files” on page 5-15](#) and [“IPX/SPX Connectivity Files” on page 5-18](#).

Initialize Disk Space

Before you initialize disk space, check the setting of your **ONCONFIG** environment variable. If you have not set it correctly, you might overwrite data from another database server. When you initialize disk space for a database server, the database server initializes the disk space specified in the current ONCONFIG configuration file.



Warning: As you create new blobspaces or dbspaces for a database server, be sure to assign each chunk to a unique location on the device. The database server does not allow you to assign more than one chunk to the same location within a single database server environment, but it remains your responsibility as administrator to make sure chunks that belong to different database servers do not overwrite each other.

Prepare Dbspace and Logical-Log Backup Environment

This section describes the effects of multiple residency on backups.

When you use multiple residency, you must maintain separate dbspace and logical-log backups for each database server instance. When you perform dbspace and logical-log backups with multiple residency, you need to be especially aware of device use.

If you can dedicate a tape drive to each database server, you can use the continuous logging option to back up your logical-log files. Otherwise, you must plan your dbspace and logical-log backup schedules carefully so that use of a device for one database server instance does not cause the other database server instance to wait. You must reset the ONCONFIG parameter each time that you switch your backup operations from one database server instance to the other.

If you use ON-Archive for your backup and restore tool, be especially aware of the following items:

- Cataloger processes
- The **config.arc** file
- The **oper_deflt.arc** file

When you use ON-Archive, each database server instance is served by its own **oncatlgr** process. When you start and stop **oncatlgr** processes with the startup and shutdown scripts (**start_oncatlgr** and **stop_oncatlgr**), the script prompts you to kill existing **oncatlgr** processes. Before you kill them, you need to know which **oncatlgr** process serves the different database server instances. The [Archive and Backup Guide](#) explains how to associate the process ID of an **oncatlgr** process with a database server instance.

To direct a database server to use a personalized copy of **config.arc** and **oper_deflt.arc**, set the **ARC_CONFIG** and **ARC_DEFAULT** environment variables to the names of the personalized configuration files. For example, suppose you copy **config.arc** and **oper_deflt.arc** to **config.mine** and **oper_deflt.mine**, respectively. You can then edit both of the **.mine** files to suit your needs. To direct a database server to use these files as your configuration files, set **ARC_CONFIG** to **config.mine** and **ARC_DEFAULT** to **oper_deflt.mine**.

Update the Operating-System Boot File

You can ask your system administrator to modify the system startup script ([“Preparing the UNIX Startup and Shutdown Scripts” on page 3-19](#)) so that each of your database server instances starts whenever the computer is rebooted (for example, after a power failure).

The startup script for a single database server instance should set the **INFORMIXDIR**, **PATH**, **ONCONFIG**, and **INFORMIXSERVER** environment variables and then execute **oninit**. To start a second instance of a database server, change the **ONCONFIG** and **INFORMIXSERVER** environment variables to point to the configuration file for the second database server and then execute **oninit** again. Do not change **INFORMIXDIR** or **PATH**.

Similarly, you can ask the system administrator to modify the shutdown script so that all instances of a database server shut down in a graceful manner.

Check INFORMIXSERVER Environment Variables for Users

If a new instance of a database should be the default database server, users need to reset the **INFORMIXSERVER** environment variable. Users might also need to update their **.informix** files.

If you use the **informix.rc** file to set environment variables for the users, you might need to update that file. The [Informix Guide to SQL: Reference](#) describes the **informix.rc** and **.informix** files.

Modes and Initialization

Chapter 8 **Managing Database Server Operating Modes**

Chapter 9 **Initializing the Database Server**

Section III



Managing Database Server Operating Modes

In This Chapter	8-3
Database Server Operating Modes	8-3
Initializing Disk Space.	8-4
Changing Database Server Operating Modes	8-5
Users Permitted to Change Modes	8-6
From Off-Line to Quiescent	8-6
From Off-Line to On-Line	8-6
From Quiescent to On-Line.	8-7
Gracefully from On-Line to Quiescent	8-7
Immediately from On-Line to Quiescent	8-8
From Any Mode Immediately to Off-Line	8-9

In This Chapter

This chapter introduces you to the database server operating modes and provides instructions on how to change the operating modes of 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 8-1](#) illustrates.

Figure 8-1
Operating Modes

Operating Mode	Description
Off-line 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.
On-line 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 on-line.

In addition, the database server can also be in one of the following modes:

- *Read-only mode* is used by the secondary database server in a high-availability data-replication 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 Off-Line to Quiescent mode.
- *Shutdown mode* is transitory. It occurs when the database server is moving from on-line to quiescent mode or from on-line (or quiescent) to off-line mode. Current users access the system, but no new users are allowed access.

Once shutdown mode is initiated, it cannot be cancelled.

Initializing Disk Space

If you are starting a new 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 on-line mode.

Operating System	Action
UNIX	■ Execute oninit -iy. ■ ON-Monitor: Select the Initialize option from the Parameters menu.
Windows NT	■ With the Services tool, 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.



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.

WIN NT

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. ♦

Changing Database Server Operating Modes

This section describes how to change from one database server operating mode to another. Each section includes information on how to change operating modes with the **oninit** and **onmode** utilities and with ON-Monitor (UNIX).

WIN NT

In Windows NT, the database server runs as a service. Windows NT 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 servername (the value of DBSERVERNAME in the ONCONFIG file). For example, the Dynamic Server service for the **newyork** database server is as follows:

```
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 NT tools, such as the NET command and the **Server Manager** tool. For more information about these methods, consult your Windows NT 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

On UNIX platforms, only users who are logged in as **root** or **informix** can change the operating mode of the database server. On Windows NT platforms, only users who are members of the **Informix-Admin** Group can change the operating mode of the database server.

From Off-Line to Quiescent

When the database server changes from off-line 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. ■ ON-Monitor: Select the Startup option from the Mode menu.
Windows NT	■ On the command line, use the starts dbservername -s command.

From Off-Line to On-Line

When you take the database server from off-line mode to on-line mode, the database server initializes shared memory. When the database server is in on-line mode, it is available to all database server sessions.

Operating System	Action
UNIX	■ Execute oninit.
Windows NT	■ With the Services tool, select the database server service and click Start. ■ On the command line, use the starts dbservername command.

From Quiescent to On-Line

When you take the database server from quiescent mode to on-line mode, all sessions gain access.

If you have already taken the database server from on-line mode to quiescent mode and you are now returning the database server to on-line mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.

Operating System	Action
UNIX	■ Execute onmode -m. ■ ON-Monitor: Select the On-Line option from the Mode menu.
Windows NT	■ With the Services tool, choose the database server service and click Continue. ■ Execute onmode -m.

Gracefully from On-Line to Quiescent

Take the database server gracefully from on-line 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 on-line to quiescent, the database server is considered to be in Shutdown mode.

Operating System	Action
UNIX	■ Execute onmode -s or onmode -sy. ■ ON-Monitor: Select the Graceful Shutdown option from the Mode menu. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work.
Windows NT	■ Execute onmode -s or onmode -sy. ■ With the Services tool, choose the database server service and click Pause.

Immediately from On-Line to Quiescent

Take the database server immediately from on-line 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 performs proper cleanup on behalf of all sessions that the database server terminated. Active transactions are rolled back.

Operating System	Action
UNIX	■ Execute onmode -u or onmode -uy. ■ ON-Monitor: Select the Immediate Shutdown option from the Mode menu. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work or until you leave the screen.
Windows NT	■ Execute onmode -u or onmode -uy.

From Any Mode Immediately to Off-Line

You can take the database server immediately from any mode to off-line mode. After you take the database server to off-line mode, reinitialize shared memory by taking the database server to quiescent or on-line mode. When you reinitialize shared memory, the database server performs a fast recovery to ensure that the data is logically consistent.

A prompt asks for confirmation to go off-line. 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 off-line mode, reinitialize shared memory by taking the database server to quiescent or on-line mode. When you reinitialize shared memory, the database server performs a fast recovery to ensure that the data is logically consistent.

The database server performs proper cleanup on behalf of all sessions that were terminated by the database server. Active transactions are rolled back.

Operating System	Action
UNIX	■ Execute onmode -k or onmode -ky The -y option eliminates the automatic prompt that confirms an immediate shutdown. ■ ON-Monitor: Select the Take Offline option from the Mode menu. ON-Monitor displays a list of all active user threads and updates it every five seconds until the last user thread completes work or until you leave the screen.
Windows NT	■ 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.

Initializing the Database Server

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

In This Chapter

Initialization of the database server refers to two related activities: disk-space initialization and shared-memory initialization. This chapter defines the two types of initialization and describes the activities that take place during initialization.

Types of Initialization

Shared-memory initialization 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.

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 recovery 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.*

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.

WIN NT

UNIX

Initializing the Database Server

On UNIX platforms, you must be logged in as **informix** or **root** to initialize the database server. On Windows NT platforms, you must be a member of the **Informix-Admin** group to initialize the database server.

The database server must be in off-line mode when you begin initialization.

You can initialize shared memory and disk space with one of the following methods:

- The **oninit** utility
For more information about **oninit**, refer to the utilities chapter in the *Administrator's Reference*.
- In Windows NT, the database server runs as a service. Use the **Service** control application to bring the database server to on-line mode.
- The **starts** command ♦
- The ON-Monitor utility ♦

For information on using these utilities to initialize the database server and change the database server mode, see [“Starting the Database Server and Initialize Disk Space” on page 3-17](#) and [Chapter 8, “Managing Database Server Operating Modes.”](#)

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 9-1 shows the main tasks completed during the two types of initialization. The following sections discuss each step.

Figure 9-1
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 oncfg_servername.servernum file.	Update oncfg_servername.servernum file.
Change to quiescent mode.	Change to quiescent mode.
Drop temporary tblspaces (optional).	

Shared-Memory Initialization	Disk Initialization
Set forced residency, if requested.	Set forced residency, if specified.
Change to on-line mode and return control to user.	Change to on-line mode and return control to user.
If the SMI tables are not current, update the tables.	Create SMI tables.

Process Configuration File

The database server uses configuration parameters to allocate shared-memory segments during initialization. If you change the size of shared memory by modifying a configuration-file parameter, you must take the database server to off-line mode and then reinitialize shared memory.

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 following file.

Operating System	ONCONFIG File
UNIX	\$INFORMIXDIR/etc/\$ONCONFIG
Windows NT	%INFORMIXDIR%\etc\%ONCONFIG%

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.

Operating System	ONCONFIG File
UNIX	\$INFORMIXDIR/etc/onconfig
Windows NT	%INFORMIXDIR%\etc\onconfig

Informix recommends 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. For more information about the configuration file, refer to [“Preparing the ONCONFIG Configuration File” on page 3-14](#).

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`. For more information about `PAGE_CONFIG`, refer to the chapter on configuration parameters in the *Administrator's Reference*. 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, buffer tables, and so on
- Virtual portion, used for most internal 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 12-25](#).

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 Structures

After the database server attaches to shared memory, it clears the shared-memory space of uninitialized data. Next the database server lays out the shared-memory header information and initializes data in the shared-memory structures. For example, 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. For more information about PAGE_PZERO, refer to the information on reserved pages in the disk structures and storage chapter of the *Administrator's Reference*.

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 10-3](#).

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 [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 23-14](#).

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 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 23-4](#).

The database server now moves to quiescent mode or on-line 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 NT. The database server uses this file during a full-system restore. For more information about the ***oncfg_servername.servernum*** file, refer to 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 14-38](#).

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

After the previous steps are complete, the database server writes an initialization complete message in the message log. For more information about the message path and the MSGPATH configuration parameter, refer to the *Administrator's Reference*.

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed in one or more of the following locations:

- The command line
- The database server message log file, specified by the **MSGPATH** environment variable
- Within ON-Monitor, if you are using it ♦

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 file. 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 on-line. 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 or ON-Archive for dbspace or logical-log 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. Informix recommends 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 11-8](#).

Disk, Memory, and Process Management

- Chapter 10 Virtual Processors and Threads
- Chapter 11 Managing Virtual Processors
- Chapter 12 Shared Memory
- Chapter 13 Managing Shared Memory
- Chapter 14 Data Storage
- Chapter 15 Managing Disk Space
- Chapter 16 Table Fragmentation and PDQ

Section IV



Virtual Processors and Threads

In This Chapter	10-3
Virtual Processors	10-3
Threads	10-4
Types of Virtual Processors.	10-6
Advantages of Virtual Processors	10-9
Sharing Processing	10-9
Saving Memory and Resources	10-10
Processing in Parallel	10-10
Adding and Dropping Virtual Processors in On-Line Mode	10-11
Binding Virtual Processors to CPUs	10-12
How Virtual Processors Service Threads	10-12
Control Structures	10-13
Context Switching.	10-14
Stacks	10-15
Queues	10-16
Ready Queues	10-17
Sleep Queues	10-17
Wait Queues	10-18
Mutexes	10-19
Virtual-Processor Classes.	10-19
CPU Virtual Processors	10-19
Determining the Number of CPU Virtual Processors Needed	10-20
Running on a Multiprocessor Computer	10-21
Running on a Single-Processor Computer	10-21
Adding and Dropping CPU Virtual Processors in On-Line Mode	10-22
Preventing Priority Aging.	10-22
Using Processor Affinity	10-23

User-Defined Classes of Virtual Processors	10-24
Purpose of User-Defined Virtual Processors.	10-25
Assigning a UDR to a User-Defined Virtual-Processor Class	10-26
Adding and Dropping User-Defined Virtual Processors in On-Line Mode	10-27
Java Virtual Processors	10-27
Disk I/O Virtual Processors	10-27
I/O Priorities	10-28
Logical-Log I/O	10-28
Physical-Log I/O	10-29
Asynchronous I/O	10-29
Network Virtual Processors.	10-32
Specifying Network Connections	10-32
Running Poll Threads on CPU or Network Virtual Processors	10-33
Number of Networking Virtual Processors Needed	10-33
Listen and Poll Threads for the Client/Server Connection	10-34
Starting Multiple Listen Threads	10-37
Communications Support Module Virtual Processor	10-39
Optical Virtual Processor	10-39
Audit Virtual Processor	10-39
Miscellaneous Virtual Processor	10-40

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 they function similarly 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.

Virtual processors are implemented as follows.

Operating System	Description of Virtual Processor
UNIX	A virtual processor is a process that the operating system schedules for processing.
Windows NT	A virtual processor is a task that Windows NT schedules as an operating-system thread.

Figure 10-1 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.

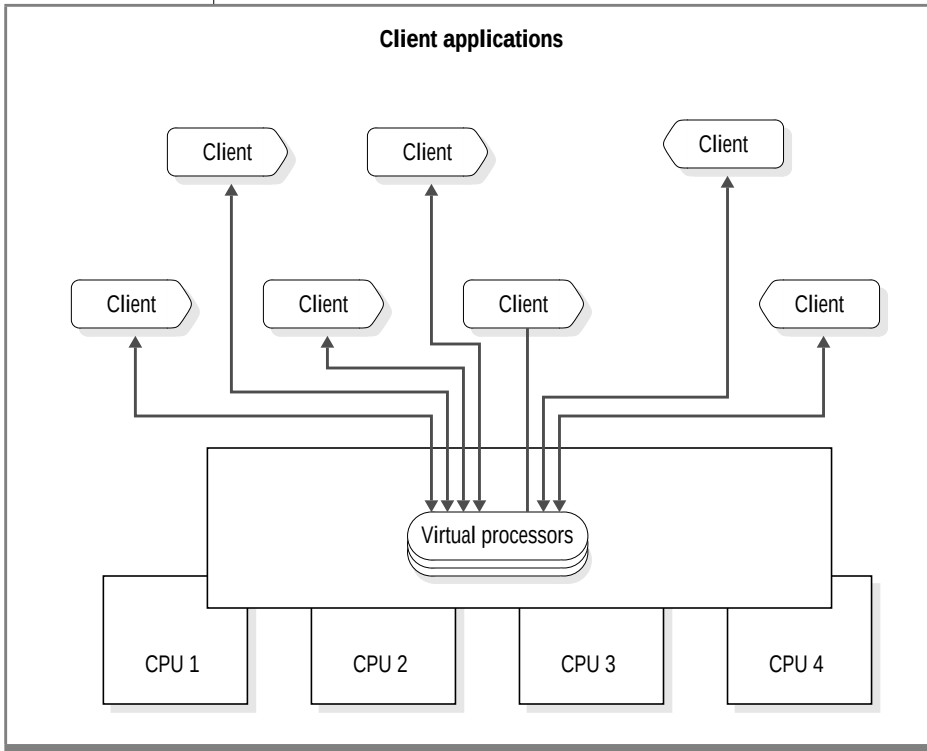


Figure 10-1
Virtual Processors

Threads

A thread is a piece of work for a virtual processor in the same way that the virtual processor is a piece of work 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 NT	A thread is a task that the virtual processor schedules internally for processing. Because the virtual processor is implemented as a Windows NT thread, database server threads run within Windows NT threads.



Important: Throughout this chapter, all references to “thread” refer to the threads created, scheduled, and destroyed by the database server. All references to “NT threads” refer to the threads created, scheduled, and destroyed by Windows NT.

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 10-10](#).

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 ON-Monitor requests, a thread to service requests from the **onmode** utility, threads for recovery, 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 10-2 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.

Figure 10-2
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 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 kernel asynchronous I/O 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	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.

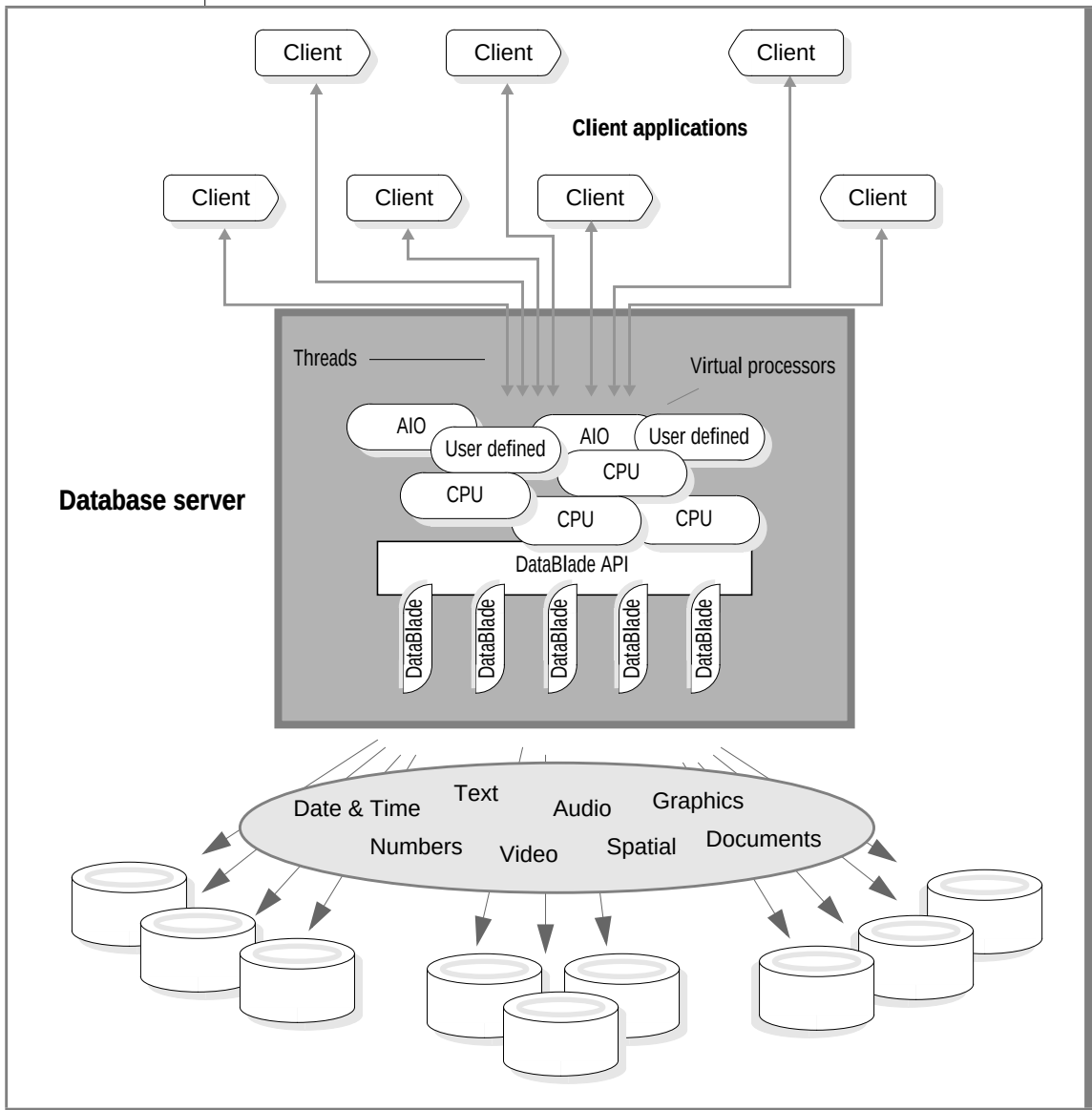
(1 of 2)

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 10-3 illustrates the major components and the extensibility of the database server.

Figure 10-3
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 10-14](#).

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 [Extending Informix Dynamic Server 2000](#).

Figure 10-4 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.

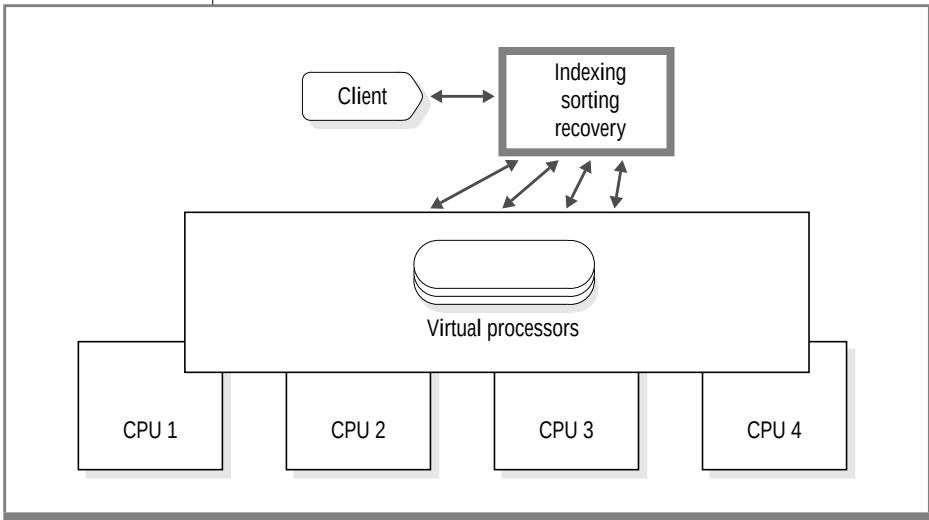


Figure 10-4
Parallel Processing

Adding and Dropping Virtual Processors in On-Line 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. While the database server is running, you can drop virtual processors of the CPU class.

You can also drop virtual processors of a user-defined class while the database server is running.

For information on how to add or drop virtual processors while the database server is in on-line mode, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#) and [“Dropping User-Defined Virtual Processors in On-Line Mode” on page 11-12](#).

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 10-23](#).

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 10-5](#) illustrates how a virtual processor accomplishes a context switch.

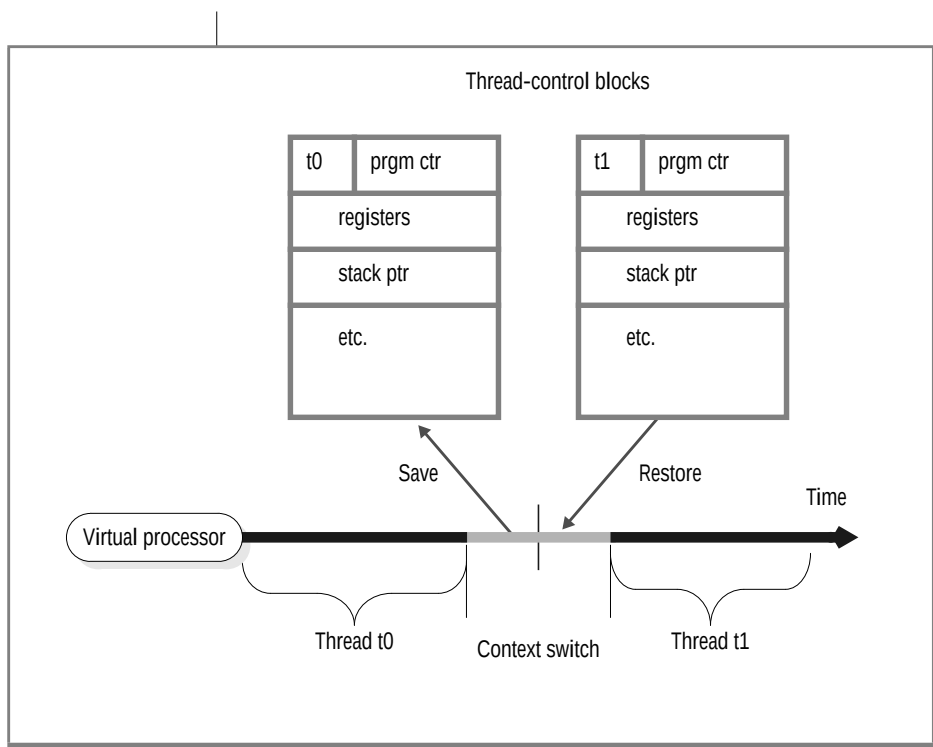


Figure 10-5
*Context Switch:
 How a Virtual
 Processor Switches
 from One Thread to
 Another*

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 12-31](#).

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 10-6](#) illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

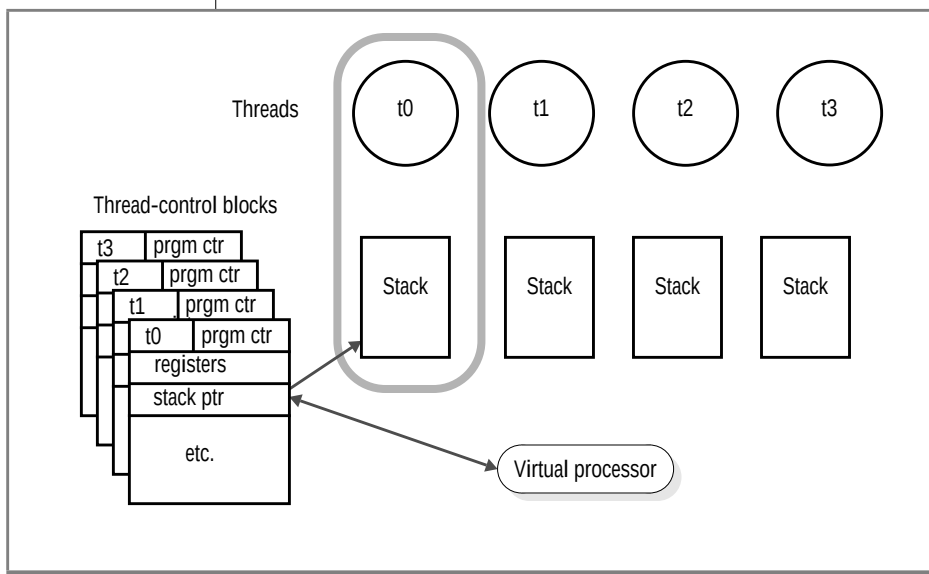


Figure 10-6
*Virtual Processors
Segregate
Nonshared Data for
Each User*

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 11-12](#). For information on how to add virtual processors while the database server is in on-line mode, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#).

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 10-7](#) illustrates how the database server threads are queued to perform database I/O.

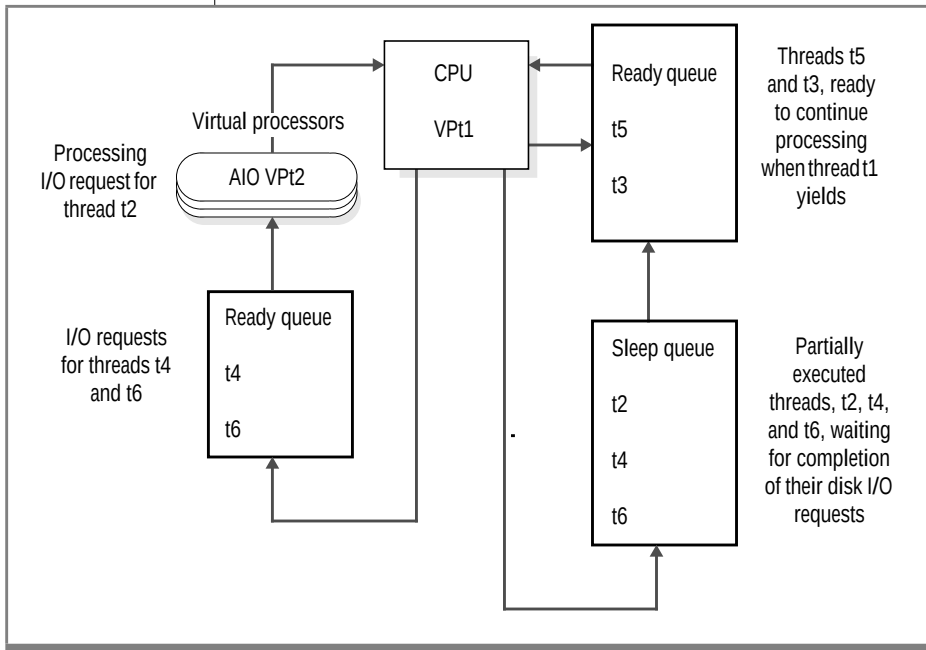


Figure 10-7
How Database Server Threads Are Queued to Perform Database I/O

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 Latches” on page 13-28](#).

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.

The NUMCPUVPS configuration parameter allows you to specify the number of CPU virtual processors that the database server starts initially. For information about the NUMCPUVPS configuration parameter, refer to the *Administrator's Reference*.

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 an additional consideration in deciding how many CPU virtual processors you need, refer to [“Running Poll Threads on CPU or Network Virtual Processors” on page 10-32](#).

You can use either the VPCLASS or the NUMCPUVPS configuration parameter to specify the number of CPU virtual processors that the database server starts initially. 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

Informix suggests 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 10-26](#).

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 only one CPU virtual processor, set the MULTIPROCESSOR configuration parameter to 0 and 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 NUMCPUVPS 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 on-line 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 On-Line Mode” on page 11-8](#).

Adding and Dropping CPU Virtual Processors in On-Line Mode

You can add or drop CPU class virtual processors while the database server is on-line. For instructions on how to do this, see [“Adding Virtual Processors in On-Line Mode” on page 11-8](#) and [“Dropping CPU Virtual Processors in On-Line Mode” on page 11-11](#).

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 [“Documentation Notes, Release Notes, Machine Notes” on page 14](#) of the Introduction.

If your operating system allows you to disable priority aging, you can disable it by setting the NOAGE parameter. For more information on the NOAGE configuration parameter, refer to the *Administrator's Reference*.

You can also disable priority aging by specifying `noage` for the priority entry in the VPCLASS configuration parameter. For more information, refer to the *Administrator's Reference*.

Using Processor Affinity

On some multiprocessor platforms that support *processor affinity*, you can assign virtual processors to specific CPUs. When you assign a virtual processor to a specific CPU, the virtual processor runs exclusively on that CPU.

You can set the following parameters in the ONCONFIG file to implement processor affinity on multiprocessor computers that support it:

- AFF_SPROC and AFF_NPROCS
- VPCLASS

Setting Processor Affinity with the AFF_SPROC and AFF_NPROCS Parameters

Set the AFF_NPROCS parameter to the number of CPUs to which you want to assign CPU virtual processors. Do not set AFF_NPROCS to a number that is less than the number of CPU virtual processors you have allocated. The number of CPUs should not be less than the number of CPU virtual processors that you allocate.

Set the AFF_SPROC parameter to the number of the first CPU to which a CPU virtual processor should be assigned. The database server assigns CPU virtual processors to CPUs in serial fashion, starting with this processor. The first CPU is number 0. For example, if the computer has four CPUs and you set NUMCPUVPS to 3, AFF_SPROC to 1, and AFF_NPROCS to 3, the three CPU virtual processors are assigned to the second, third, and fourth CPUs, respectively.

The value of AFF_NPROCS plus the value of AFF_SPROC must be less than or equal to the number of CPUs. In the previous example, if you set AFF_SPROC to 2, the database server would display an error message because 3 (AFF_NPROCS) plus 2 (AFF_SPROC) equals 5, and only four CPUs are available.

Figure 10-8 illustrates the concept of processor affinity.

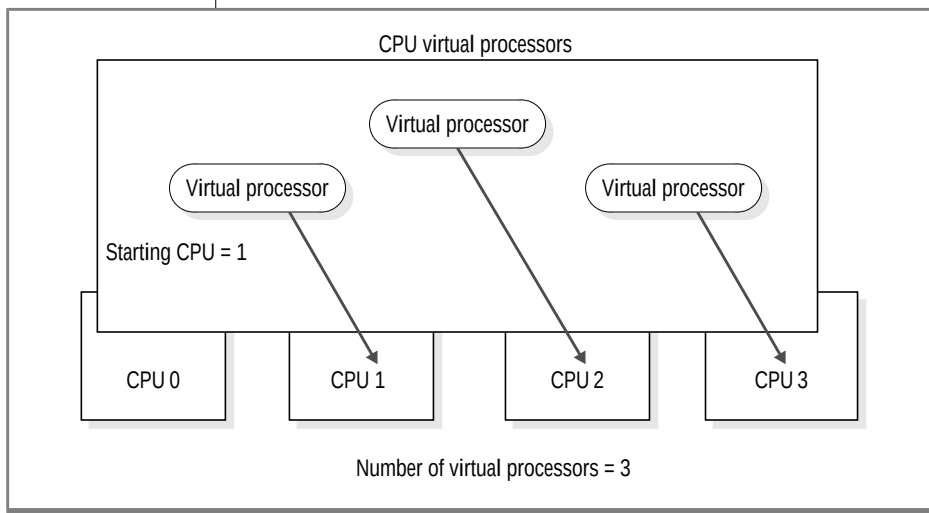


Figure 10-8
Processor Affinity

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. In the following example, the affinity entry assigns a virtual processor to CPUs 1, 2, and 3.

```
VPCLASS CPU,num=3,aff=1-3
```

For more information about VPCLASS, refer to 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 you execute a C-language UDR in a user-defined VP instead of in the CPU VP
- How to assign a C UDR to a particular user-defined VP class
- How to add and drop user-defined VPs when the database server is in on-line mode

Purpose of User-Defined Virtual Processors

The purpose of providing user-defined classes of virtual processors is to protect the database server from *ill-behaved* user-defined routines. An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

A well-behaved C UDR has none of these characteristics. Execute only well-behaved C 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 UDR because these UDRs do not execute in CPU virtual processors. For a more detailed discussion of ill-behaved routines, refer to the [DataBlade API Programmer's Manual](#).

The VPCLASS parameter defines a user-defined VP class. For more information, see [“Specifying VPCLASS” on page 11-5](#) and the configuration 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, **GreaterThanOrEqualTo()**, and specifies that calls to this routine should be executed by the user-defined VP class named UDR:

```
CREATE FUNCTION GreaterThanOrEqualTo(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 **GreaterThanOrEqualTo** 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 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 [Informix Guide to SQL: Syntax](#).

Adding and Dropping User-Defined Virtual Processors in On-Line Mode

You can add or drop virtual processors in a user-defined class while the database server is on-line. For instructions on how to do this, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#) and [“Dropping CPU Virtual Processors in On-Line Mode” on page 11-11](#).

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. Multiple JVPs can exist in the same database server to allow parallel invocations of Java UDRs.

For more information about UDRs written in Java, refer to [Creating UDRs in Java](#).

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 kernel-asynchronous I/O.

On operating systems that do not support kernel-asynchronous I/O, 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 kernel-asynchronous I/O (KAIO) when it is available on a platform. If the database server implements kernel-asynchronous I/O, 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 kernel-asynchronous I/O, refer to the machine notes file. ♦

For more information about nonlogging I/O, refer to [“Asynchronous I/O” on page 10-29](#).

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 10-9](#) shows.

Figure 10-9
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:

- Kernel asynchronous I/O is not implemented.
- The logical-log files are in cooked disk space.

Only when kernel asynchronous I/O 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:

- Kernel asynchronous I/O is not implemented.
- The physical-log file is stored in buffered-file chunks.

Only when kernel asynchronous I/O 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 23-4](#).) 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
- Kernel-asynchronous I/O (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 kernel-asynchronous I/O when the following conditions exist:

- The computer and operating system support it.
- A performance gain is realized.
- The I/O is to raw disk space.

The database server implements kernel-asynchronous I/O 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.

UNIX

Informix implements kernel-asynchronous I/O when it ports the database server to a platform that supports this feature. The database server administrator does not configure kernel-asynchronous I/O. To see if kernel-asynchronous I/O is supported on your platform, see the machine notes file. ♦

AIO Virtual Processors

If the platform does not support kernel-asynchronous I/O 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 either VPCLASS or NUMAIOVPS to specify the number of AIO virtual processors that the database server starts initially. Informix recommends that you use VPCLASS instead of NUMAIOVPS. You cannot use both NUMAIOVPS and VPCLASS AIO in the same ONCONFIG file. For information about VPCLASS and NUMAIOVPS, 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 on-line mode. For more information, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#).

You cannot drop AIO virtual processors while the database server is in on-line 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 11-12](#).

If the database server implements kernel-asynchronous I/O 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 kernel-asynchronous I/O, 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 kernel-asynchronous I/O 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 5, “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 [“If the sqlhosts file has multiple entries with the same dbservername, only the first one is used. ®”](#) on page 5-28.

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 *in-line* 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 on-line mode, you cannot drop a CPU virtual processor that is running a poll thread.

Number of Networking Virtual Processors Needed

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.

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 10-10](#) causes the database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Figure 10-10
A Listen Thread for Each Listen Port

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1

The listen thread opens the port and requests one of the poll threads for the specified interface / protocol combination to monitor the port for client requests. The poll thread runs either in the CPU virtual processor or in the network virtual processor for the connection that is being used. For information on the number of poll threads, refer to [“Number of Networking Virtual Processors Needed”](#) on page 10-33.

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 10-32 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 10-11](#) illustrates the roles of the listen and poll threads in establishing a connection with a client application.

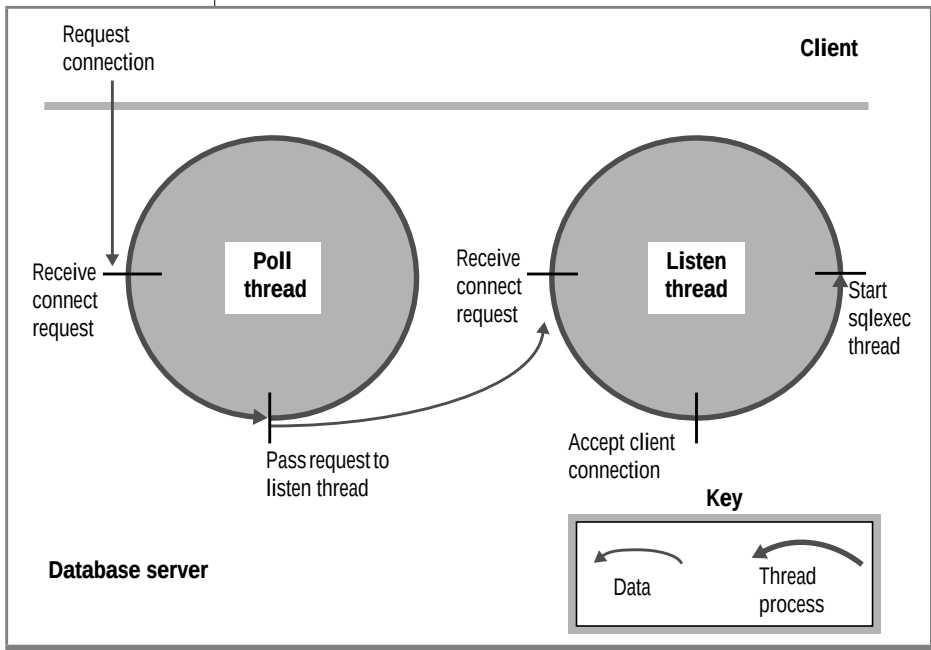


Figure 10-11
The Roles of the Poll and the Listen Threads in Connecting to a Client

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.

For a shared-memory connection, the poll thread places the message in the communications portion of shared memory. ♦

UNIX

Figure 10-12 illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

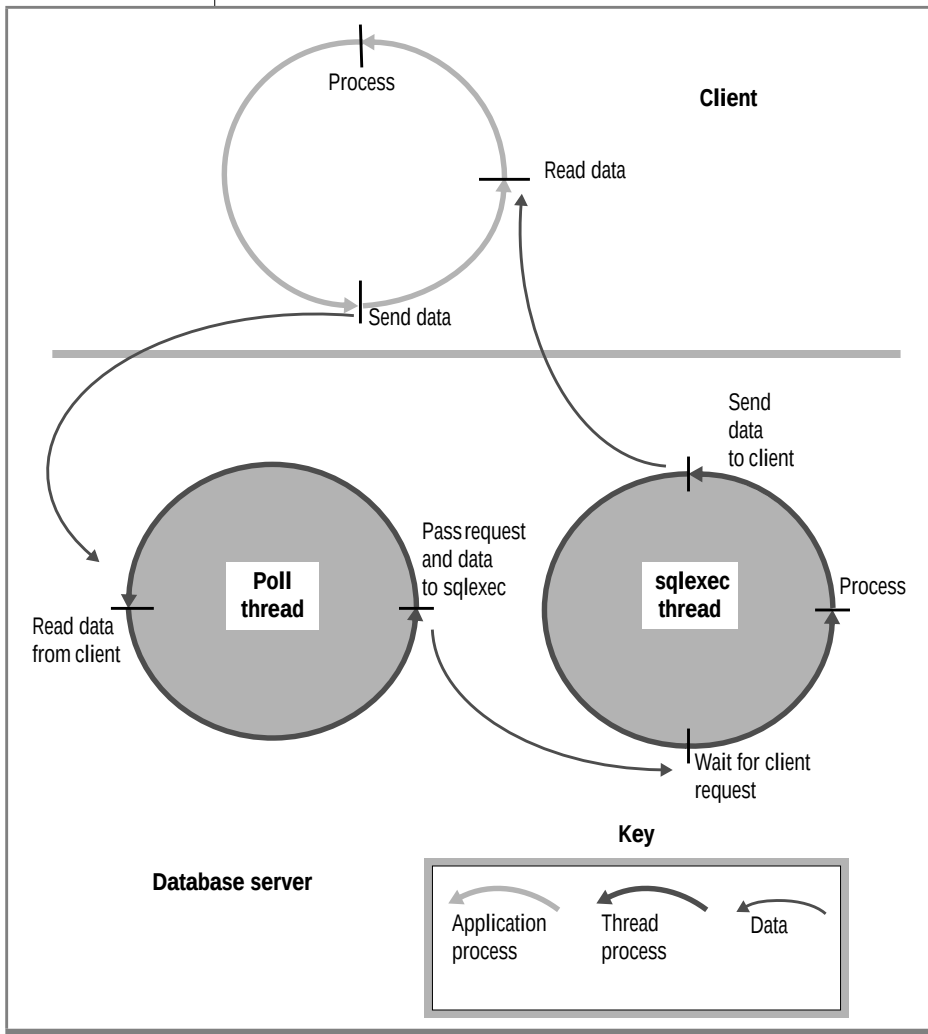


Figure 10-12
The Roles of the Poll
and sqlexec Threads
in Communicating
with the Client
Application

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 10-13](#) 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 10-13
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 10-14 on page 10-38](#) cause the database server to start three listen threads for the **onsoctcp** interface / protocol combination, one for each of the ports defined.

Figure 10-14

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 10-14](#), 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 5-14](#).

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 10-13](#), the **sqlhosts** file or registry entries shown in [Figure 10-15 on page 10-39](#) cause the database server to start three listen threads for the same interface / protocol combination (as did the entries in [Figure 10-14](#)). 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 10-15

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 5, “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 [Guide to the Optical Subsystem](#).

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 [Trusted Facility Manual](#).

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	11-3
Setting Virtual-Processor Configuration Parameters	11-3
Setting Virtual-Processor Configuration Parameters with a Text Editor	11-4
Disabling Priority Aging	11-5
Specifying VPCLASS	11-5
Setting Virtual-Processor Configuration Parameters with ON-Monitor	11-6
Starting and Stopping Virtual Processors	11-8
Adding Virtual Processors in On-Line Mode.	11-8
Adding Virtual Processors in On-Line Mode with onmode . . .	11-9
Adding Virtual Processors in On-Line Mode with ON-Monitor .	11-9
Adding Network Virtual Processors	11-11
Dropping CPU Virtual Processors in On-Line Mode	11-11
Dropping User-Defined Virtual Processors in On-Line Mode . . .	11-12
Monitoring Virtual Processors	11-12
Monitoring Virtual Processors with Command-Line Utilities . . .	11-12
Monitoring Virtual Processors with SMI Tables.	11-15

In This Chapter

This chapter describes how to set the configuration parameters that affect database server virtual processors. This chapter also tells you 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 10, “Virtual Processors and Threads.”](#)

Setting Virtual-Processor Configuration Parameters

As **root** or user **informix**, you can set the configuration parameters for the database server virtual processors with the following tools:

- A text editor
- ON-Monitor

ON-Monitor does not allow you to change all configuration parameters. This chapter describes the configuration parameters that you can change with ON-Monitor and explains how to change them. ♦

To implement any changes that you make to configuration parameters, you must reinitialize shared memory. For information on how to reinitialize shared memory, refer to [“Reinitializing Shared Memory” on page 13-16.](#)

For more information on configuration parameters, refer to the *Administrator’s Reference*.

Setting Virtual-Processor Configuration 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 11-1 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 10-19](#).

Figure 11-1
ONCONFIG Parameters for Configuring Virtual Processors

Parameter	Purpose
AFF_NPROCS	Specifies the number of CPUs to which CPU virtual processors will be assigned (multiprocessor computers only)
AFF_SPROC	Specifies the first CPU (of AFF_NPROCS) to which a CPU virtual processor will be assigned
MULTIPROCESSOR	Specifies that you are running on a multiprocessor computer
NETTYPE	Specifies parameters for network protocol threads (and virtual processors)
NOAGE	Specifies no priority aging of processes by the operating system
NUMAIOVPS	Specifies the number of AIO virtual processors
NUMCPUVPS	Specifies the number of CPU virtual processors
SINGLE_CPU_VP	Specifies that you are running a single CPU virtual processor
VPCLASS	Specifies a class of virtual processors, the number that the database server should start, and the maximum number allowed

UNIX

Disabling Priority Aging

Use the NOAGE parameter to disable process priority aging on platforms that allow this feature.

For recommended values for these database server parameters on UNIX platforms, refer to your machine notes file.

Specifying VPCLASS

The VPCLASS configuration parameter consolidates information from the NUMCPUVPS, NUMAIOVPS, AFF_SPROC, and AFF_NPROCS parameters. Informix suggests that you use VPCLASS instead of those 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.



Important: You cannot set VPCLASS with ON-Monitor; you can set VPCLASS only with a text editor.

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, see the *Administrator's Reference*, [“User-Defined Classes of Virtual Processors” on page 10-24](#), and [“Java Virtual Processors” on page 10-27](#).

Setting Virtual-Processor Configuration Parameters with ON-Monitor

To set the virtual-processor configuration parameters with ON-Monitor, select Parameters from the main menu and then select the perFormance option.

Figure 11-2 shows only the perFormance screen entries for configuring virtual processors. For each entry, it shows the name of the associated parameter in the ONCONFIG file within a pair of brackets ([]).

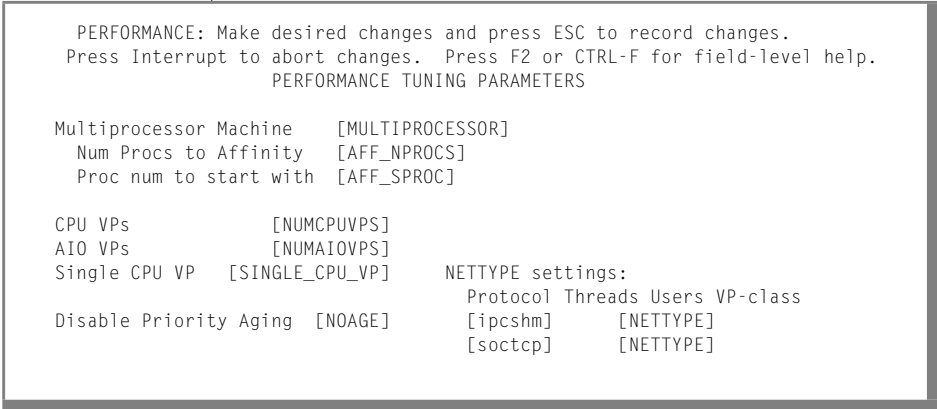


Figure 11-2
*Partial View of
ON-Monitor
PerFormance
Screen with the
ONCONFIG
Parameter for the
Virtual-Processor
Entries*

Figure 11-3 shows the full perFormance screen; the shaded entries set configuration parameters for virtual processors.

Figure 11-3
ON-Monitor Performance Screen

PERFORMANCE: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.

PERFORMANCE TUNING PARAMETERS

Multiprocessor Machine	[N]
Num Procs to Affinity	[0]
Proc num to start with	[0]
CPU VPs	[5]
AIO VPs	[1]
Single CPU VP	[N]
Use OS Time	[N]
Disable Priority Aging	[N]
Off-Line Recovery Threads	[10]
On-Line Recovery Threads	[1]
Num of LRUS queues	[8]

LRU Max Dirty	[60]
LRU Min Dirty	[50]
Checkpoint Interval	[300]
Num of Read Ahead Pages	[50]
Read Ahead Threshold	[20]

NETTYPE settings:

Protocol	Threads	Users	VP-class
[ipcshm]	[2]	[5]	[CPU]
[soctcp]	[2]	[5]	[]

Are you running on a multiprocessor machine?

Each row of entries under NETTYPE settings describes a separate NETTYPE parameter, one for each of the protocols available on the computer. The four columns for these entries (**Protocol**, **Threads**, **Users**, and **VP-class**) correspond to the four fields of the NETTYPE parameter.

For more information on the ONCONFIG parameters that are associated with the database server virtual processors, refer to [Figure 11-1 on page 11-4](#).

Starting and Stopping Virtual Processors

When you start the **oninit** process to start the database server, **oninit** 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 10-19](#).

The database server allows you to start a maximum of 1000 virtual processors.

Once the database server is in on-line mode, you can start additional virtual processors to improve performance, if necessary. For more information, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#).

While the database server is in on-line mode, you can drop only virtual processors of the CPU class. For more information, refer to [“Dropping CPU Virtual Processors in On-Line Mode” on page 11-11](#).

To terminate the database server and thereby terminate all virtual processors, use the **-k** option of the **onmode** utility. For more information on using **onmode -k**, refer to the utilities chapter of the *Administrator's Reference*.

Adding Virtual Processors in On-Line Mode

While the database server is in on-line mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, STR, TLI, and SOC.

You can start these additional virtual processors with one of the following two methods:

- The **-p** option of the **onmode** utility
- ON-Monitor ♦

You can also start additional virtual processors for user-designated classes to run user-defined routines. The user-defined class must first be defined on a VPCLASS parameter. For more information about user-defined virtual processors, refer to [“Assigning a UDR to a User-Defined Virtual-Processor Class” on page 10-26](#).

You can use only the **onmode** utility to start additional virtual processors for a user-defined class. For the format of the **onmode** command, refer to the *Administrator's Reference*.

Adding Virtual Processors in On-Line Mode with onmode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in on-line mode. Specify the number of virtual processors that you want to add with a positive number that is greater than the number of virtual processors that are currently running. 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 On-Line Mode with ON-Monitor

To use ON-Monitor to add virtual processors while the database server is in on-line mode, select Modes from the main menu, and then select Add-Proc.

[Figure 11-4](#) shows the ON-Monitor Add-Proc screen, which allows you to add virtual processors in the following classes: CPU, AIO, LIO, PIO, and network.

```
DD VIRTUAL PROCESSORS: Press ESC to add virtual processors.  
Press Interrupt to cancel and return to the modes menu.
```

ADDING VIRTUAL PROCESSORS

```
Number of CPU Virtual Processors to add          [  0 ]  
Number of Asynchronous IO Virtual Processors to add [  0 ]  
Number of Logical log IO Virtual Processors to add [  0 ]  
Number of Physical log IO Virtual Processors to add [  0 ]  
Number of Network Virtual Processors to add      [  ] [  ]  
                                                  [  ] [  ]  
                                                  [  ] [  ]
```

```
Enter the number of asynchronous IO processors to add
```

Figure 11-4
*ON-Monitor Add-
Proc Screen*

The logical-log and physical-log entries on the Add-Proc screen allow you to enter a number greater than 2, but the database server does not start more than two virtual processors in either of these classes. The database server automatically starts one virtual processor in each of these classes unless mirroring is used, in which case it starts two.

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-designated class. For information on adding virtual processors of a user-designated class in on-line mode, refer to [“Adding Virtual Processors in On-Line Mode with onmode”](#) on page 11-9.

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 on-line 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 Virtual Processors in On-Line Mode

While the database server is in on-line mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, 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 on CPU virtual processors and poll threads, refer to [“Running Poll Threads on CPU or Network Virtual Processors” on page 10-33](#).

Dropping User-Defined Virtual Processors in On-Line Mode

While the database server is in on-line mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of a user-defined class. 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
```

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 10-24](#).

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.

Monitoring Virtual Processors with Command-Line Utilities

You can use the following **onstat -g** options to monitor virtual processors:

- **glo**
- **ioq**
- **rea**

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 11-5](#) shows an example of the output from this option.

Figure 11-5
onstat -g glo Output

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

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**. You can also see the maximum queue length (since the database server started) in the **maxlen** column. Each chunk serviced by the AIO virtual processors has one line in the **onstat -g ioq** output, identified by the **gfd** queue name. You can correlate the line in **onstat -g ioq** with the actual chunk because the chunks are in the same order as in the **onstat -d** output. For example, in the **onstat -g ioq** output in [Figure 11-6](#), there are two **gfd** queues. The first **gfd** queue holds requests for **root_chunk** because it corresponds to the first chunk shown in the **onstat -d** output in [Figure 11-6](#). Likewise, the second **gfd** queue holds requests for **chunk1** because it corresponds to the second chunk in the **onstat -d** output.

If the database server has a mixture of raw devices and cooked files, the **gfd** queues correspond only to the cooked files in **onstat -d** output.

```
onstat -g iioq

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

smoke% onstat -d
Dbspaces
address number  flags    fchunk  nchunks  flags    owner    name
alde1d8  1           1        1        1        N        informix rootdbs
aldf550  2           1        2        1        N        informix space1
  2 active, 2047 maximum
Chunks
address  chk/dbs offset  size    free    bpages  flags  pathname
alde320  1   1   0     75000   66447   P0-    /ix/root_chunk
aldf698  2   2   0       500     447    P0-    /ix/chunk1
  2 active, 2047 maximum
```

Figure 11-6
*onstat -g iioq and
onstat -d Output*

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.

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 of those virtual processors to your configuration. [Figure 11-7](#) 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     sqlexec
```

Figure 11-7
onstat -g rea Output

Monitoring Virtual Processors with SMI Tables

Query the **sysvpprof** 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	12-5
Shared Memory	12-5
Shared-Memory Use	12-6
Shared-Memory Allocation.	12-7
Shared-Memory Size	12-9
Action to Take If SHMTOTAL Is Exceeded	12-10
Processes That Attach to Shared Memory	12-11
How a Client Attaches to the Communications Portion	12-11
How Utilities Attach to Shared Memory	12-12
How Virtual Processors Attach to Shared Memory	12-12
Defining a Unique Key Value	12-13
Specifying Where to Attach the First Shared-Memory Segment	12-14
Attaching Additional Shared-Memory Segments	12-15
Defining the Shared-Memory Lower-Boundary Address	12-15
Resident Shared-Memory Segments	12-17
Resident Portion of Shared Memory	12-18
Shared-Memory Header.	12-18
Shared-Memory Buffer Pool	12-18
Buffer Overflow to the Virtual Portion	12-20
Buffer Size	12-20
Memory-Resident Tables	12-20
Logical-Log Buffer.	12-21
Physical-Log Buffer	12-23
High-Availability Data-Replication Buffer	12-23
Lock Table	12-24

Virtual Portion of Shared Memory	12-25
Management of the Virtual Portion of Shared Memory	12-25
Size of the Virtual Portion of Shared Memory	12-25
Components of the Virtual Portion of Shared Memory	12-26
Shared-Memory Internal Tables	12-26
Big Buffers	12-30
Session Data	12-30
Thread Data.	12-30
Dictionary Cache	12-32
SQL Statement Cache	12-32
Sorting Memory	12-32
SPL Routine and the UDR Cache	12-33
Global Pool	12-33
Communications Portion of Shared Memory	12-33
Virtual-Extension Portion of Shared Memory	12-34
Concurrency Control	12-34
Shared-Memory Mutexes	12-35
Shared-Memory Buffer Locks	12-35
Types of Buffer Locks	12-35
Database Server Thread Access to Shared Buffers	12-36
LRU Queues	12-36
Components of LRU Queue	12-36
Pages in Least-Recently Used Order	12-37
LRU Queues and Buffer-Pool Management	12-37
Number of LRU Queues to Configure.	12-38
Number of Cleaners to Allocate	12-39
Number of Pages Added to the MLRU Queues	12-39
End of MLRU Cleaning.	12-40
Configuring the Database Server to Read Ahead	12-41
Database Server Thread Access to Buffer Pages	12-42
Identify the Page	12-42
Determine the Level of Lock Access	12-42
Try to Locate the Page in Shared Memory	12-43
Locate a Buffer and Read Page from Disk	12-43
Lock the Buffer If Necessary	12-43
Release the Buffer Lock and Wake a Waiting Thread	12-44

Flushing Data to Disk	.12-45
Events That Prompt Flushing of Buffer-Pool Buffers.	.12-46
Flushing Before-Images First.	.12-46
Flushing the Physical-Log Buffer	.12-46
Events That Prompt Flushing of the Physical-Log Buffer	.12-47
When the Physical-Log Buffer Becomes Full	.12-48
Synchronizing Buffer Flushing	.12-49
Ensuring That Physical-Log Buffers Are Flushed First.	.12-49
Flushing the Shared-Memory Pool Buffer	.12-50
Describing Flushing Activity.	.12-50
Foreground Write	.12-51
LRU Write	.12-51
Chunk Write	.12-52
Flushing the Logical-Log Buffer.	.12-52
When the Logical-Log Buffer Becomes Full	.12-53
After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging	.12-54
When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates	.12-54
When a Checkpoint Occurs	.12-54
When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File.	.12-55
Buffering Large-Object Data	.12-55
Writing Simple Large Objects	.12-55
Blobpages and Shared Memory	.12-55
Creation of Simple Large Objects	.12-56
Creation of Blobpage Buffers	.12-56
Accessing Smart Large Objects	.12-58
Memory Use on 64-Bit Platforms.	.12-59

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 13, “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 uses 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 12-1 illustrates the shared-memory scheme.

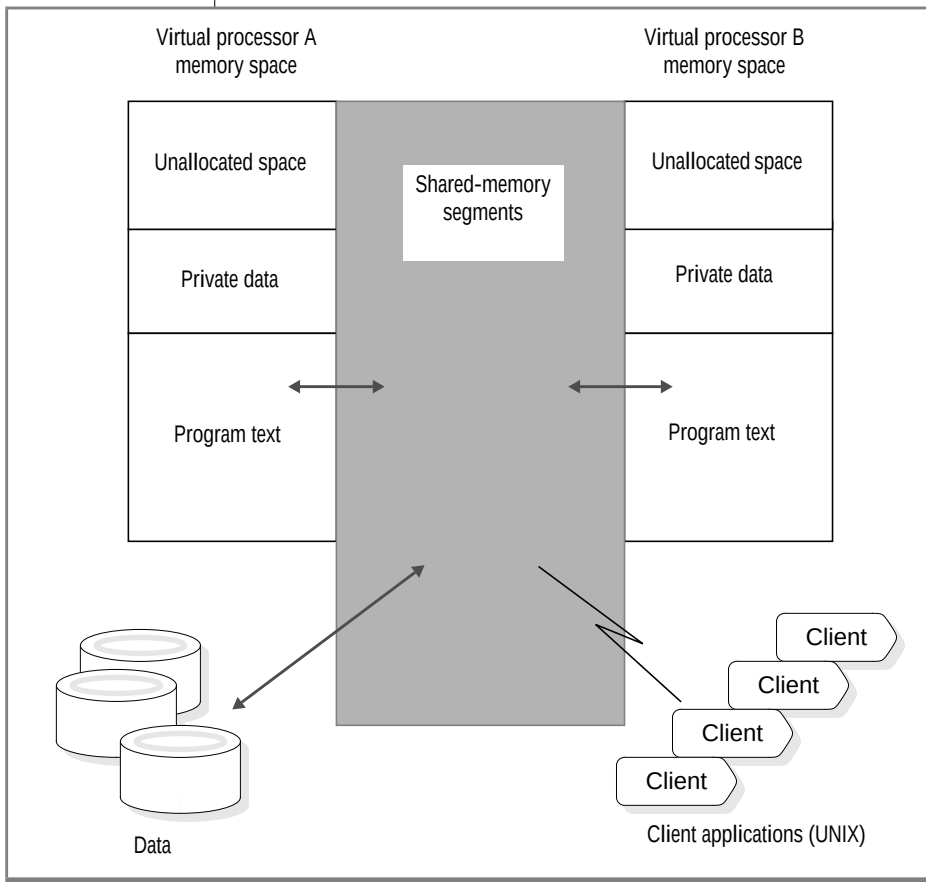


Figure 12-1
How the Database Server Uses Shared Memory

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 ♦
- The *virtual-extension* portion

Each portion of shared memory consumes one or more operating-system segments. When the database server initializes shared memory, it allocates at least two operating-system segments, one for the resident portion and one for the virtual portion. It might allocate more segments if the maximum segment size is not large enough.

Depending upon operating system settings for shared memory, the database server may allocate one segment for both the resident portion and virtual portion.

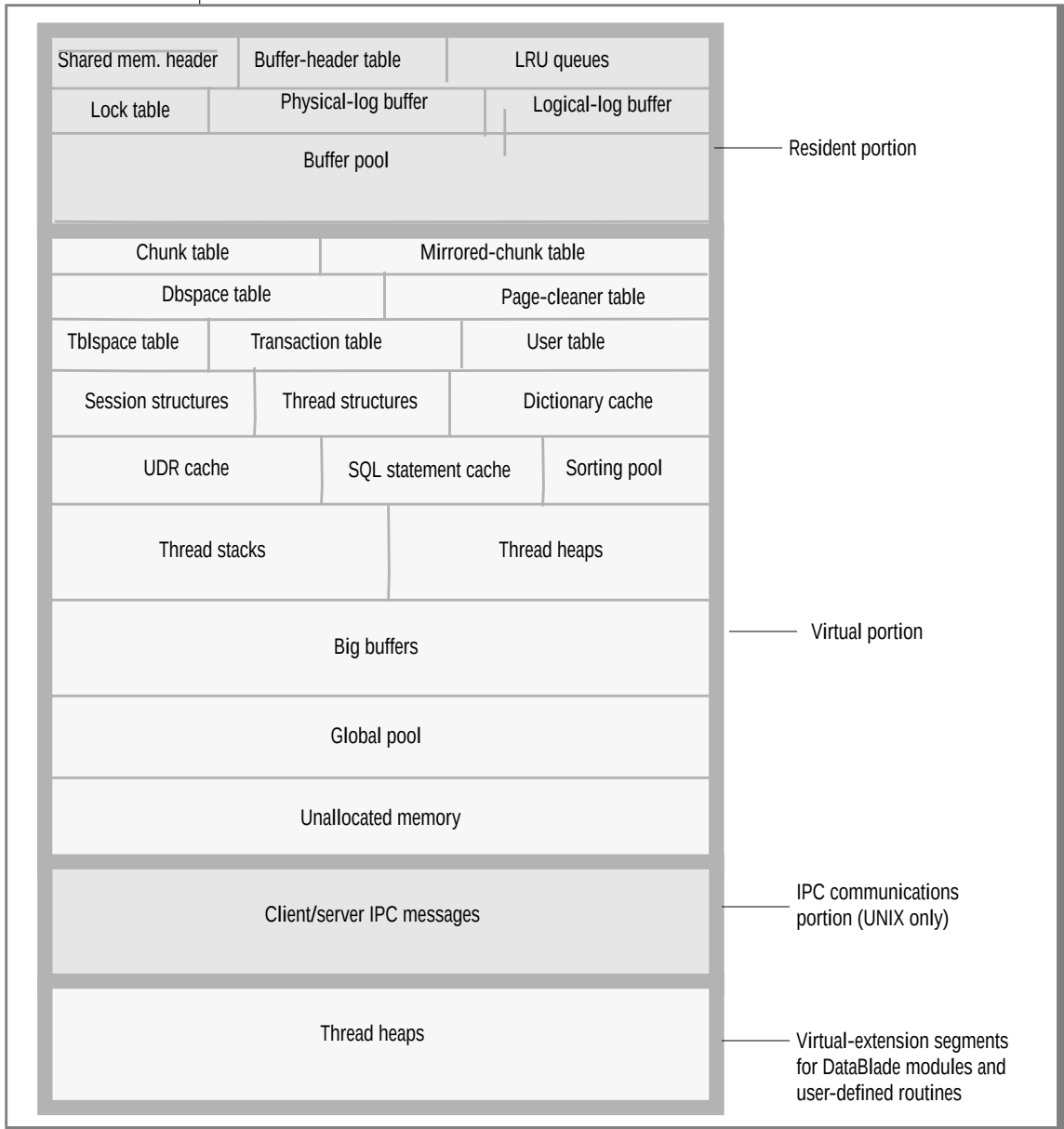
UNIX

If the **sqlhosts** file specifies shared-memory communications, the database server allocates memory for the communications portion. ♦

The database server adds operating-system segments as needed to the virtual and virtual-extension portions of shared memory. [Figure 12-2 on page 12-8](#) 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 off-line mode to quiescent mode or from off-line mode directly to on-line mode. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple threads. For more information about modes, refer to [“Managing Database Server Operating Modes” on page 8-3](#).

Figure 12-2
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 8 kilobytes in size and managed by a bit map.

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 the **-g seg** option of **onstat** 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.

Action to Take If SHMTOTAL Is Exceeded

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.

UNIX

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

The following sections describe how each type of process attaches to the database server shared memory.

UNIX

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 5, “Client/Server Communications,”](#) and [“Network Virtual Processors”](#) on page 10-32.

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 [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 NT	%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, **SERVENUM** and **SHMBASE**, to meet these requirements.

When a virtual processor attaches to shared memory, it performs the following major steps:

1. Accesses the SERVERNUM parameter from the ONCONFIG file
2. Uses SERVERNUM to calculate a shared-memory key value
3. 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.
4. Directs the operating system to attach the first shared-memory segment to its process space at SHMBASE
5. 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.

Defining a Unique Key Value

The database server uses the ONCONFIG parameter SERVERNUM to calculate a unique key value for its shared-memory segments. All virtual processors within a single database server instance share the same key value. When each virtual processor attaches to shared memory, it calculates the key value as follows:

$$(\text{SERVERNUM} * 65536) + \text{shmkey}$$

The value of *shmkey* is set internally and cannot be changed by the user. (The *shmkey* value is 52564801 in hexadecimal representation or 1,381,386,241 in decimal.) The value (SERVERNUM * 65,536) is the same as multiplying SERVERNUM by hexadecimal 10,000.

When more than one database server instance exists on a single computer, the difference in the key values for any two instances is the difference between the two SERVERNUM values, multiplied by 65,536.

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: *Informix recommends 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.
- 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. ♦

UNIX

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 12-3 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 13-6.

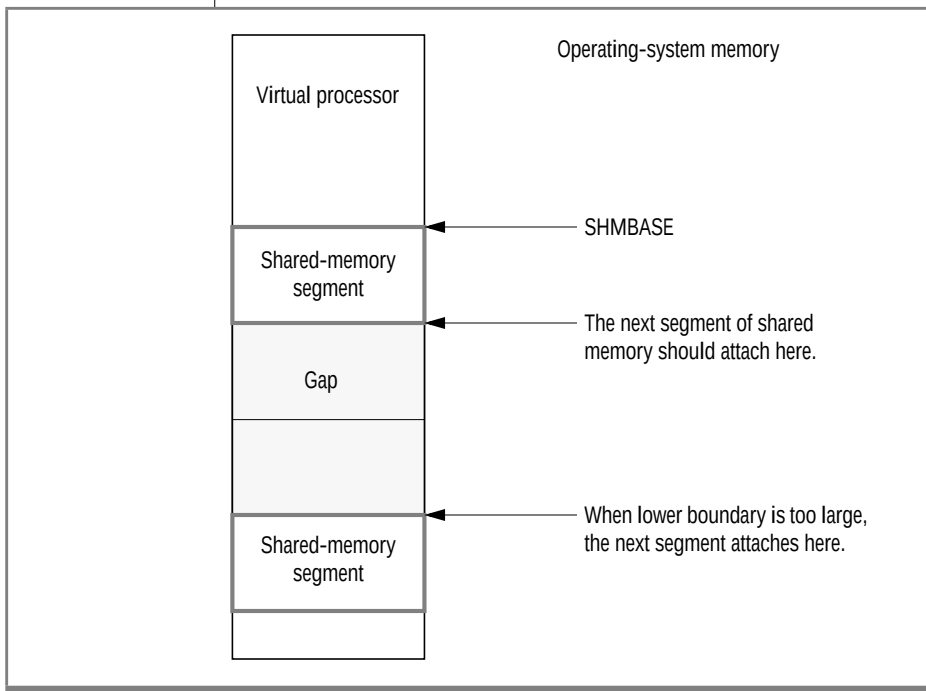


Figure 12-3
*Shared-Memory
Lower-Boundary
Address Overview*

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 one kilobyte, 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 12-4 illustrates the shared-memory header and the buffer pool.

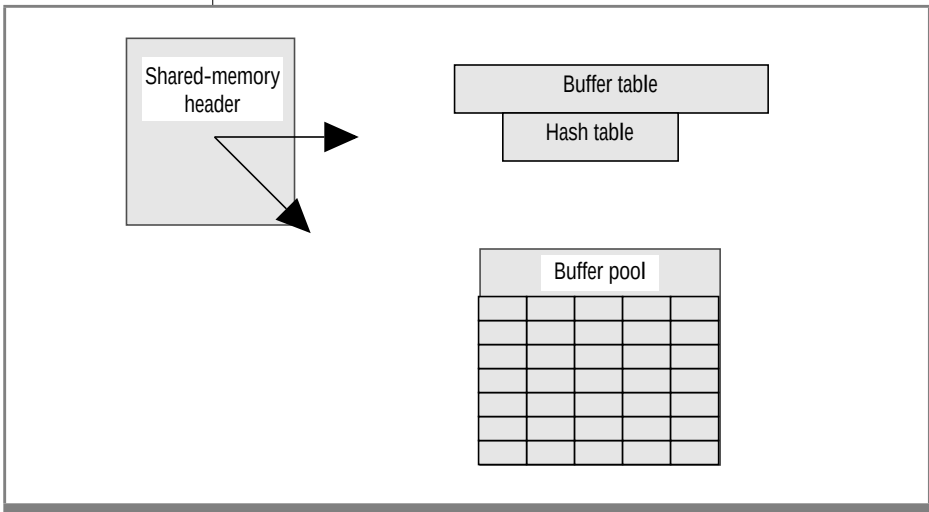


Figure 12-4
Shared-Memory
Buffer Pool

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 12-35](#). For a description of mutexes, refer to “[Mutexes](#)” on [page 10-19](#).

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 12-30](#) and [“Creation of Blobpage Buffers” on page 12-55](#).) 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](#).

The **-b** option of the **onstat** utility displays information about the buffers. For information on **onstat**, refer to the utilities chapter in the *Administrator's Reference*.

UNIX

On UNIX, you also can determine the page size for your system by choosing the Parameters menu, Shared-memory option in ON-Monitor. ON-Monitor displays a list of shared-memory parameters. The database server page size is the last entry on the page. ♦

Memory-Resident Tables

You can specify that buffers containing one or more fragments of a table or index remain resident in shared memory as much as possible. Any pages that belong to a resident fragment, either currently in the buffer or loaded in the future, are considered last for replacement when a free buffer is requested.

Use the SET TABLE and SET INDEX statements to turn on the resident state for one or more fragments of a table or index. For information on the SET TABLE and SET INDEX statements, refer to the [Informix Guide to SQL: Syntax](#).

You must be the database system administrator (DBSA), user **informix** on UNIX, or a member of the **Informix-Admin** group on Windows NT to turn on or change the resident state. As the DBSA, you can create a DBA-privileged SPL routine to grant the privilege to other users.



Important: *The resident state of a table fragment is not persistent; it must be specified each time that you start the server. The default state is nonresident.*

If you add new fragments to a resident table, the new fragments are not automatically marked as resident. You must use `SET TABLE` or `SET INDEX` to declare the new fragments as resident or redeclare the table as resident. If you use the `SET TABLE` or `SET INDEX` statement to turn residency off, or if you use the `ALTER FRAGMENT` statement to drop a fragment, its residency is cleared.

When you turn on residency, the buffer pool is scanned, and any buffers that belong to these fragments are marked as the highest priority. When you turn off residency, the buffer pool is scanned, and all buffers belonging to that partition are put back into the buffer pool with the default priority. When the fragment is dropped, the resident state of all the buffers for that partition in the buffer pool is cleared and put back in the buffer pool with the lowest priority.

The DBSA must keep track of how many tables are marked resident. If the buffer pool contains too many resident pages, the caching effect is defeated. The number of resident pages that can be held in memory is exceeded, and high-priority pages are swapped to disk and replaced with other high-priority pages.

You can use the **-p** and **-P** options of the **onstat** utility to determine the rate of buffer pool usage and to optimize the use of memory resident tables, respectively. The **onstat -t** option enables you to determine which tablespaces are resident. 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 12-5](#).

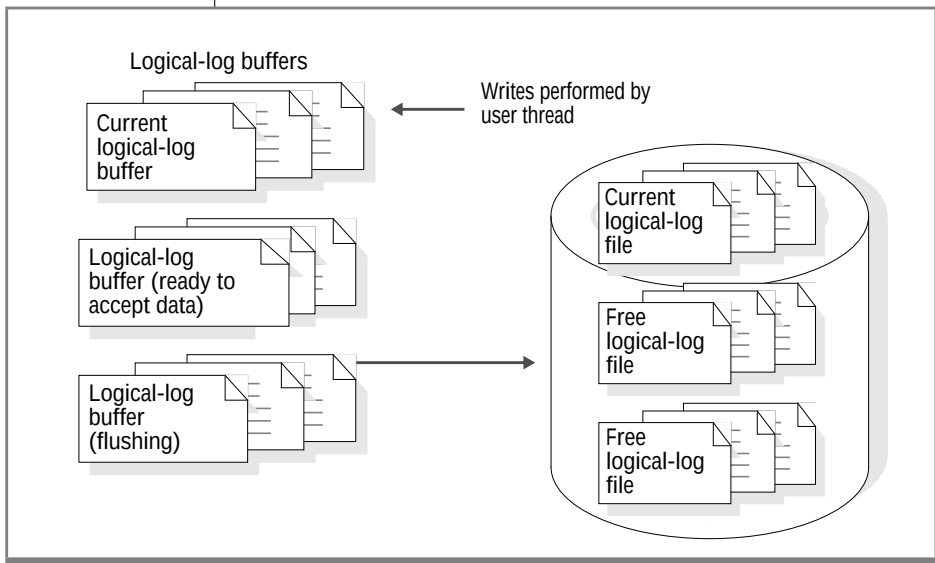


Figure 12-5
The Logical-Log Buffer and Its Relation to the Logical-Log Files on Disk

For a description of how the database server flushes the logical-log buffer, refer to [“Flushing the Logical-Log Buffer” on page 12-51](#).

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 12-54](#).

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 12-46](#). For information on monitoring the physical-log file, refer to [“Monitoring Physical and Logical Logging Activity” on page 22-6](#).

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 requires two instances of the database server, a primary instance and a secondary instance, running on two computers. If you implement high-availability data replication for your database server, the database server holds logical-log records in the high-availability data-replication buffers before it sends them to the secondary database server. A high-availability data-replication 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 12-21](#). For more information on how the data-replication buffer is used, refer to [“How High-Availability Data Replication Works” on page 27-8](#).

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 entry is one lock. A single transaction can own multiple locks. For an explanation of locking and the SQL statements associated with locking, refer to the [Informix Guide to SQL: Tutorial](#). For information on performance considerations for locking, refer to your [Performance Guide](#).

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 number of entries in the lock table, set the LOCKS configuration parameter. However, the lock table grows dynamically if more locks are needed. For information on specifying the number of locks available to sessions, refer to the chapter on configuration parameters in the *Administrator's Reference*. For information on how to estimate the number of locks, refer to the chapter on configuration effects on memory utilization in your [Performance Guide](#).

For information on byte-range locks, how the lock table grows dynamically and monitoring locks, refer to the chapter on locking 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 portion of shared memory, set the SHMADD parameter in the ONCONFIG file.

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 13-18](#) and to the 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)
- 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 14-11](#).

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, or 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 12-18](#). For information on how to monitor the buffers, refer to [“Monitoring Buffers” on page 13-21](#).

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; off-line, on-line, or recovery mode; and whether this chunk is part of a blobspace. For information on monitoring chunks, refer to [“Monitoring Chunks” on page 15-46](#).

The maximum number of entries in the chunk table might be limited by the maximum number of file descriptors that your operating system allows per process. You can usually specify the number of file descriptors per process with an operating-system kernel-configuration parameter. For details, consult your operating-system manuals.

Dbspace Table

The dbspace table tracks storage spaces in the database server. The dbspace-table information includes the following information about each dbspace:

- Dbspace number
- Dbspace name and owner
- Dbspace mirror status (mirrored or not)
- Date and time that the dbspace was created

If the storage space is a blobspace, flags indicate the media where the blobspace is located: magnetic, removable, or optical media. If the storage space is an sbpace, it contains internal tables that track smart-large-object metadata and large contiguous blocks of pages containing user data.

For information on monitoring dbspaces, refer to [“Monitoring the Database Server for Disabling I/O Errors” on page 15-44](#).

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 15-53](#).

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 [Informix Guide to SQL: Tutorial](#), the [Informix Guide to SQL: Reference](#), and the [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 cleaner thread, and one or more 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.

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 10-14](#).

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 10-15](#). 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 *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.

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 database server uses the SQL statement cache to store parsed and optimized SQL statements that a user executes. When other users execute a statement stored in the SQL statement cache, the database server does not parse and optimize the statement again, which might improve performance. For more information on the SQL statement cache, refer to your [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.

UNIX

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 12-11](#).

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 in the output from the monitoring tools), refer to [“Monitoring Latches” on page 13-28](#).

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

Each buffer in the buffer pool is tracked through several linked lists of pointers to the buffer table. A buffer holds data for the purpose of caching. These linked lists are the least-recently used (LRU) queues.

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 LRUS, combined with the LRU_MIN_DIRTY and LRU_MAX_DIRTY parameters, to control how frequently the shared-memory buffers are flushed to disk.

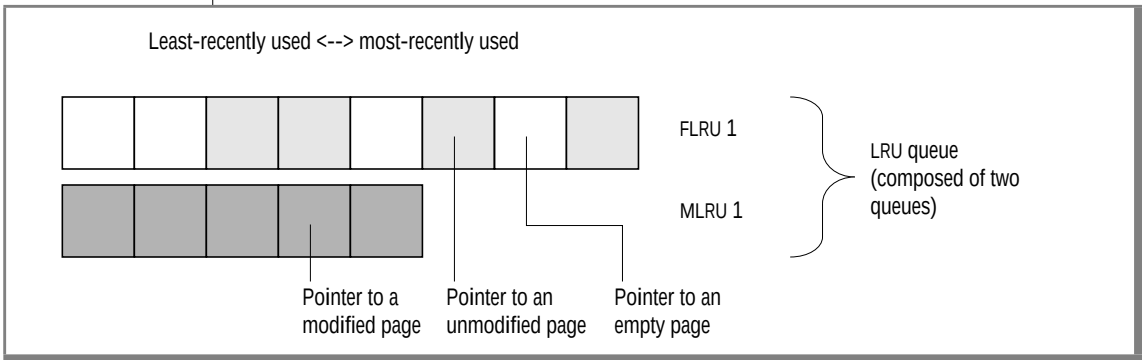
Components of LRU Queue

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 12-6](#) illustrates the structure of the LRU queues.

Figure 12-6
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) by the number of LRU queues (LRUS).

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 13-25](#).

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.

Informix recommends initial values for the LRUS configuration parameter based on the number of CPUs that are available on your computer. If your computer is a uniprocessor, start by setting LRUS 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 LRUS, 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, Informix recommends 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 can occur if you do not 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 much more dirty 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.

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, 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 12-44](#).

[Figure 12-7](#) 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

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 12-7
*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, 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 12-8](#) 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
```

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: $1000 \times .050 = 50$

Figure 12-8
*How
LRU_MIN_DIRTY
Specifies the Point
at Which Page
Cleaning Can End*

For more information on how the database server flushes the buffer pool, refer to [“Flushing Data to Disk” on page 12-44](#).

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 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” on page 13-20](#) 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 12-35.](#))

The process of accessing a data buffer consists of the following steps:

1. Identify the data requested by physical page number.
2. Determine the level of lock access needed by the thread for the requested buffer.
3. Attempt to locate the page in shared memory.
4. If the page is not in shared memory, locate a buffer in an FLRU queue, and read the page in from disk. If the page is in shared memory, proceed with step 5.
5. Proceed with processing, locking the buffer if necessary.
6. When finished accessing the buffer, release the lock.
7. Wake waiting threads with compatible lock-access types, if any exist.

Identify the Page

The database server threads request a specific data row, and the database server searches for the page that contains the row.

Determine the Level of Lock Access

Next the database server determines the requested level of lock access: share or exclusive.

Try to Locate the Page in Shared Memory

The thread first attempts to locate the requested page in shared memory. To do this, it acquires a mutex on the hash table that is associated with the buffer table. Then it searches the hash table to see if an entry matches the requested page. If the thread finds an entry for the page, it releases the mutex on the hash table and tries to acquire the mutex on the buffer entry in the buffer table.

The thread tests the current lock-access level of the buffer. If the levels are compatible, the requesting thread gains access to the buffer and sets its own lock. If the current lock-access level is incompatible, the requesting thread puts itself in the wait queue for the buffer.

The buffer state, unmodified or modified, is irrelevant to locking; even unmodified buffers can be locked.

If you configure the database server to use read-ahead, the database server performs a read-ahead request when the number of pages specified by the RA_THRESHOLD parameter remain to be processed in memory.

Locate a Buffer and Read Page from Disk

If the requested page must be read from disk, the thread first locates a usable buffer in the FLRU queues. The database server selects an FLRU queue at random and tries to acquire the mutex associated with the queue. If the mutex can be acquired, the buffer at the least-recently used end of the queue is used. If another thread holds the mutex, the first thread tries to acquire the mutex of another FLRU queue.

If you configure the database server to use read-ahead, the database server reads the number of pages specified by the RA_PAGES configuration parameter.

Lock the Buffer If Necessary

After a usable buffer is found, the buffer is temporarily removed from the FLRU queue. The thread creates an entry in the shared-memory buffer table as the page is read from disk into the buffer.

Release the Buffer Lock and Wake a Waiting Thread

When the thread is finished with the buffer, it releases the buffer lock. If any threads are waiting for the buffer, it wakes one up. However, this procedure varies, depending on whether the releasing thread modified the buffer.

When the Buffer Is Not Modified

If a thread does not modify the data, it releases the buffer as unmodified.

The release of the buffer occurs in steps. First, the releasing thread acquires the mutex on the buffer table that enables it to modify the buffer entry.

Next, it checks if other threads are sleeping, waiting for this buffer. If so, the releasing thread wakes the first thread in the wait queue that has a compatible lock-access type. The waiting threads are queued according to priorities that encompass more than just *first-come, first-served* hierarchies. (Otherwise, for example, threads waiting for exclusive access could wait forever.)

If no thread in the wait queue has a compatible lock-access type, any thread waiting for that buffer can receive access.

If no thread is waiting for the buffer, the releasing thread tries to release the buffer to the FLRU queue where it was found. If the latch for that FLRU queue is unavailable, the thread tries to acquire a latch for a randomly selected FLRU queue. When the FLRU queue latch is acquired, the unmodified buffer is linked to the most-recently used end of the queue.

After the buffer is returned to the FLRU queue, or the next thread in the wait queue is awakened, the releasing thread removes itself from the user list for the buffer and decrements the shared-user count by one.

When the Buffer Is Modified

If the thread intends to modify the buffer, to update a row in a table, for example, it acquires the mutex for the buffer and changes the buffer lock-access type to exclusive.

In most cases, a copy of the before-image of the page is needed for data consistency. If necessary, the thread determines whether a before-image of this page was written to either the physical-log buffer or the physical log since the last checkpoint. If not, a copy of the page is written to the physical-log buffer. Then the data in the page buffer is modified. If any transaction records are required for logging, those records are written to the logical-log buffer.

After the mutex for the buffer is released, the thread is ready to release the buffer. First, the releasing thread acquires the mutex on the buffer table that enables it to modify the buffer entry. Next, the releasing thread updates the time stamp in the buffer header so that the time stamp on the buffer page and the time stamp in the header match. Statistics describing the number and types of writes performed by this thread are updated.

The lock is released as described in the previous section, but the buffer is appended to the MLRU queue associated with the original FLRU queue.

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 12-46](#).
- Logical-log buffer
See [“Flushing the Logical-Log Buffer” on page 12-51](#).

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.

Events That Prompt Flushing of 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. (See [“Flushing the Shared-Memory Pool Buffer” on page 12-49.](#))
- 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 23-4.](#))

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. Writing the before-image to the physical log buffer and then flushing the buffer page to disk is illustrated in [Figure 12-9.](#)

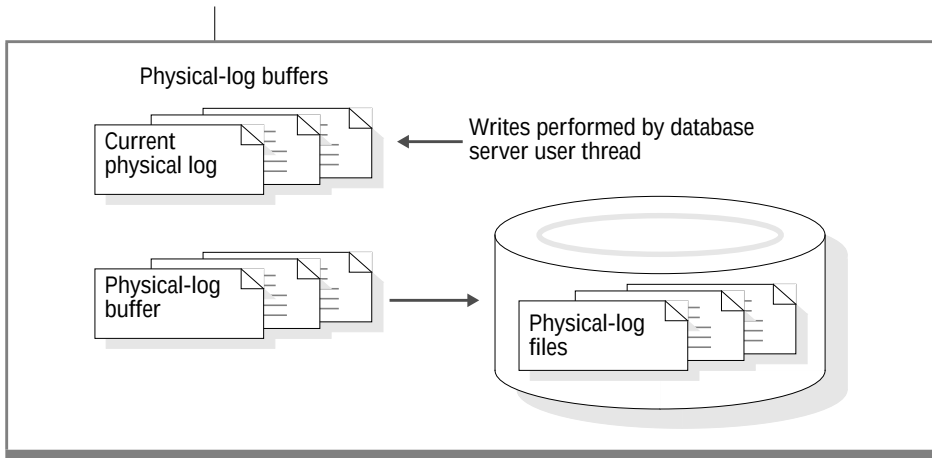


Figure 12-9
The Physical-Log Buffer and Its Relation to the Physical Log on Disk

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 21, “Physical Logging.”](#) For a description of checkpoints and fast recovery, refer to [Chapter 23, “Checkpoints and Fast Recovery.”](#)

Events That Prompt Flushing of the Physical-Log Buffer

The following four 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.

When the Physical-Log Buffer Becomes Full

Buffer flushing that results from the physical-log buffer becoming full proceeds as follows.

When a user thread needs to write a before-image to the physical-log buffer, it acquires the mutex associated with the physical-log buffer and the mutex associated with the physical log on disk. If another thread is writing to the buffer, the incoming thread must wait for the mutexes to be released.

After the incoming thread acquires the mutexes, but before the write, the thread checks to see what percent of the physical log is full.

If the Log Is More Than 75 Percent Full

If the physical log is more than 75 percent full, the thread sets a flag to request a fuzzy checkpoint. Next, the thread claims the amount of space in the buffer that it needs for its write and releases the buffer mutex so that other threads can access the buffer. Finally, it copies the data to the space that it claimed in the buffer. The checkpoint does not begin until all user threads, including this one, are out of critical sections. For more information on critical sections and checkpoints, refer to [“How the Database Server Achieves Data Consistency” on page 23-3](#).

If the Log Is Less Than 75 Percent Full

If the physical log is less than 75 percent full, the thread compares the page counter in the physical-log buffer header to the buffer capacity. If this one-page write does not fill the physical-log buffer, the thread reserves space in the log buffer for the write and releases the mutex. Any thread waiting to write to the buffer is awakened. After the thread releases the mutex, it writes the page to the reserved space in the physical-log buffer. The sequence of this operation increases concurrency and eliminates the need to hold the mutex during the write.

If this one-page write fills the physical-log buffer, flushing is initiated. First the page is written to the current physical-log buffer, filling it. Next, the thread latches the other physical-log buffer. The thread switches the shared-memory current-buffer pointer, making the newly latched buffer the current buffer. The mutex on the physical log on disk and the mutex on this new, current buffer are released, which permits other user threads to begin writing to the new current buffer. Last, the full buffer is flushed to disk, and the mutex on the buffer is released.

Each write to the physical-log buffer writes one page.

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.

Ensuring That Physical-Log Buffers Are Flushed First

When page cleaning is initiated on the shared-memory buffer pool, the page-cleaner thread must coordinate the flushing so that the physical-log buffer is flushed first. Time-stamp comparison determines the order.

The database server stores a time stamp each time that the physical-log buffer is flushed. If a page-cleaner thread needs to flush a page in a shared-memory buffer, the page cleaner compares the time stamp in the modified buffer with the time stamp that indicates the point when the physical-log buffer was last flushed.

If the time stamp on the page in the buffer pool is equal to or more recent than the time stamp for the physical-log buffer flush, the before-image of this page conceivably could be contained in the physical-log buffer. In this case, the physical-log buffer must be flushed before the shared-memory buffer pages are flushed.

Flushing the Shared-Memory Pool Buffer

After the physical-log buffer is flushed, the user thread updates the time stamp in shared memory that describes the most-recent physical-log buffer flush. The specific page in the shared-memory buffer pool that is marked for flushing is now flushed. The number of modified buffers in the queue is compared to the value of `LRU_MIN_DIRTY`. If the number of modified buffers is greater than the value represented by `LRU_MIN_DIRTY`, another page buffer is marked for flushing. The time-stamp comparison is repeated. If required, the physical-log buffer is flushed again.

When no more buffer flushing is required, the page-cleaner threads sleep until buffer flushing is required again, and they are awakened to do the work. (For more information, refer to [“Sleep Queues” on page 10-17.](#)) You can tune the page-cleaning parameters (`LRU_MIN_DIRTY` and `LRU_MAX_DIRTY`) to influence the frequency of buffer flushing. For a description of how these parameters determine when page cleaning begins and ends, refer to [“LRU Queues” on page 12-35.](#)

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 13-25.](#)

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 12-35.](#))

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 19, “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.

When the Logical-Log Buffer Becomes Full

When a user thread needs to write records to the logical-log buffer, it acquires the mutexes associated with the logical-log buffer and the current logical log on disk. If another thread is writing to the buffer, the incoming thread must wait for the mutexes to be released.

After the incoming thread acquires the mutexes, but before the write, the thread checks how much logical-log space is available on disk. When the logical-log space on disk is full, and the database server switches to a new logical log, it checks if the percent of used log space is greater than the long-transaction high-water mark, specified by the LTXHWM parameter in the ONCONFIG file. For a description of this configuration parameter and information on specifying a value for it, refer to the *Administrator's Reference*.

If no long-transaction condition exists, the logical-log I/O thread compares the available space in the logical-log buffer with the size of the record to be written. If the write does not fill the logical-log buffer, the thread writes the record, releases latches, and awakens any threads that are waiting to write to the buffer.

If the write fills the logical-log buffer, flushing is initiated as follows:

1. The thread latches the next logical-log buffer. The thread then switches the shared-memory current-buffer pointer, making the newly latched buffer the current buffer.
2. The thread writes the new record to the new current buffer. The thread releases the latch on the logical log on disk and the latch on this current buffer, permitting other logical-log I/O threads to begin writing to this buffer.
3. The full logical-log buffer is flushed to disk, and the latch on the buffer is released. This logical-log buffer is now available for reuse.

After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging

If a transaction is prepared or terminated in a database with unbuffered logging, the logical-log buffer is immediately flushed. Flushing might cause a waste of some disk space. Typically, many logical-log records are stored on a single page. However, because the logical-log buffer is flushed in whole pages, the whole page is flushed even if only one transaction record is stored on the page. In the worst case, a single COMMIT logical-log record (COMMIT WORK) could occupy a page on disk, and all remaining space on the page would be unused. However, the cost in disk space of using unbuffered logging is minor compared to the benefits of insured data consistency.

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 [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 23-4](#).

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 12-44](#).

You can also assign simple large objects to a blobspace. The database server writes simple large objects to a blobspace differently than 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 [“Blobpage Logging” on page 19-32](#).

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 blobpage 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 12-10](#) illustrates the process of writing a simple large object to a blobpage.

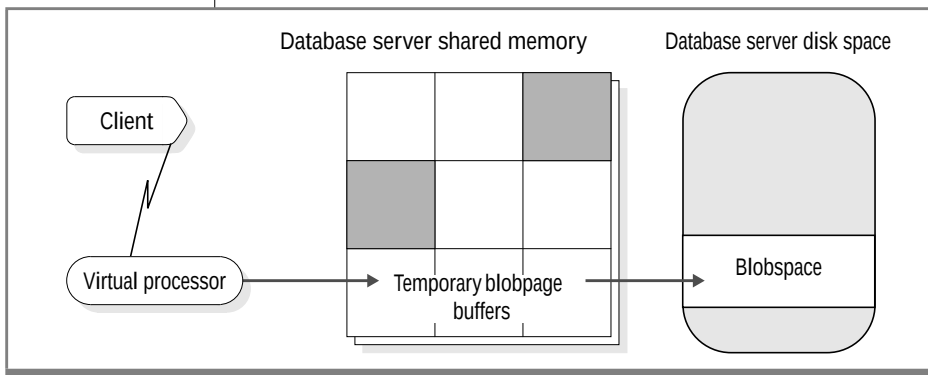


Figure 12-10
Writing Simple Large Object to a Blobpage

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 sbspace. You cannot store simple large objects in an sbspace, and you cannot store smart large objects in a blobspace. An sbspace consists of a user-data area and a metadata area. The user-data area contains the smart-large-object data. The metadata area contains information about the content of the sbspace. For more information on sbspaces, refer to [“Sbspaces” on page 14-24](#).

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

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

Managing Shared Memory

In This Chapter	13-3
Setting Operating-System Shared-Memory Configuration Parameters	13-3
Maximum Shared-Memory Segment Size.	13-4
Increasing the Amount of Virtual Memory	13-5
Using More Than Two Gigabytes of Memory	13-5
Maximum Number of Shared-Memory Identifiers.	13-6
Shared-Memory Lower-Boundary Address	13-6
Maximum Amount of Shared Memory for One Process	13-7
Semaphores	13-7
Setting Database Server Shared-Memory Configuration Parameters	13-8
Setting Parameters for Resident Shared Memory with ON-Monitor	13-9
Setting Parameters for Resident Shared Memory with a Text Editor.	13-11
Setting Parameters for Virtual Shared Memory with ON-Monitor	13-12
Setting Parameters for Virtual Shared Memory with a Text Editor	13-13
Setting Parameters for Shared-Memory Performance Options with ON-Monitor.	13-14
Setting Parameters for Shared-Memory Performance Options with a Text Editor	13-15
Reinitializing Shared Memory	13-16
Turning Residency On or Off for Resident Shared Memory	13-17
Turning Residency On or Off in On-Line Mode.	13-17
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory	13-18

Adding a Segment to the Virtual Portion of Shared Memory.	13-18
Monitoring Shared Memory	13-19
Monitoring Shared-Memory Segments	13-19
Monitoring the Shared-Memory Profile	13-20
Using Command-Line Utilities	13-20
Using ON-Monitor	13-20
Using SMI Tables	13-21
Monitoring Buffers	13-21
Using Command-Line Utilities	13-21
Using ON-Monitor	13-24
Using SMI Tables	13-24
Monitoring Buffer-Pool Activity	13-25
Using Command-Line Utilities	13-25
Using SMI Tables	13-27
Monitoring Latches	13-28
Using Command-Line Utilities	13-28
Using ON-Monitor	13-29
Using SMI Tables	13-29

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 12, “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 [“Documentation Notes, Release Notes, Machine Notes”](#) on page 14 in 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.

WIN NT

Increasing the Amount of Virtual Memory

Insufficient memory for Dynamic 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.

WIN NT

Using More Than Two Gigabytes of Memory

The database server can access shared-memory segments larger than two gigabytes on Windows NT systems. However, you must enable this feature with an entry in the Windows NT 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 NT without additional drivers.

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 12-12](#).

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.

Shared-Memory Lower-Boundary Address

When the database server attaches shared-memory segments subsequent to the first segment, it assumes that the segment can be attached contiguous with the previous one; that is, that a segment can be attached at the address of the previous segment plus the size of that segment. However, your operating system might set a parameter that defines a lower-boundary address for attaching shared-memory segments. If the size of a segment would cause it to cross the lower-boundary address, the segment is attached at a point beyond the end of the previous segment, creating a gap between shared-memory segments. For an illustration of this situation, refer to [“How Virtual Processors Attach to Shared Memory” on page 12-12](#).

WIN NT

The default load address for a dynamic link library (DLL) on Windows NT 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.

If Dynamic Server 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 for Dynamic Server, run **onstat -g seg** and view the total memory. ♦

Maximum Amount of Shared Memory for One Process

Check that the maximum amount of memory that can be allocated for one process is equal to the total addressable shared-memory size for a single operating-system process. The following equation expresses the concept another way:

$$\text{Maximum amount of shared memory for one process} = (\text{Maximum number of attached shared-memory segments per process}) \times (\text{Maximum shared-memory segment size})$$

If this relationship does not hold, one of two undesirable situations could develop:

- If the total amount of shared memory is less than the total addressable shared-memory size, you can address more shared memory for the operating system than is available.
- If the total amount of shared memory is greater than the total addressable size of shared memory, you can never address some amount of shared memory that is available. That is, space that could potentially be used as shared memory cannot be allocated.

UNIX

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 (**ipcs** 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
- Using ON-Monitor ♦

On UNIX, you must be **root** or user **informix** to use either method. On Windows NT, you must be a user in the **Informix Admin** group.

Before any changes that you make to the configuration parameters take effect, you must reinitialize shared memory by stopping and starting the database server.

UNIX

UNIX

Setting Parameters for Resident Shared Memory with ON-Monitor

To set the configuration parameters for the resident portion of shared memory with ON-Monitor, select the Shared-Memory option from the Parameters menu. [Figure 13-1](#) shows an example of a Shared-Memory screen. The shaded entries set configuration parameters for the resident portion of shared memory.

Figure 13-1
ON-Monitor
Shared-Memory
Screen

```

SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                SHARED MEMORY PARAMETERS
Server Number   [ 0]          Server Name [odyssey_01]
Server Aliases [oddsoc_011,oddsoc_012,oddsoc_013]
Dbospace Temp   [
Deadlock Timeout [ 0] Secs   Number of Page Cleaners [ 1]
Forced Residency [Y]         Stack Size (Kbytes) [ 32]
Non Res. SegSize (Kbytes) [ 4000] Optical Cache Size (Kbytes)[ 0]

Physical Log Buffer Size [ 32] Kbytes   Dbospace Down Option [ 0]
Logical Log Buffer Size [ 32] Kbytes   Preserve Log for Log Backup [N]
Max # of Logical Logs [ 6]            Transaction Timeout [ 300]
Max # of Locks [ 2000]                Long TX HWM [ 50]
Max # of Buffers [ 200]                Long TX HWM Exclusive [ 60]
Index Page Fill Factor [ 90]
Add SegSize (Kbytes) [ 8192]
Total Memory (Kbytes) [ 0]

Shared memory size [ 546] Kbytes      Page Size [ 2] Kbytes

Enter a unique value to be associated with this version of Informix database server

```



Important: The configuration parameters SHMADD and SHMTOTAL are described with the parameters that affect the resident portion of shared memory, but they affect both the resident and virtual portions of shared memory.

Figure 13-2 shows an example of only the Shared-Memory screen entries that affect the configuration of the resident portion of shared memory. For each entry, it shows within brackets ([]) the name of the associated parameter in the ONCONFIG file.

```

SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
      SHARED MEMORY PARAMETERS
Server Number      [SERVER_NUM]

                                Number of Page Cleaners[CLEANERS]
Forced Residency   [Y]

Physical Log Buffer Size [ PHYSBUFF] Kbytes
Logical Log Buffer Size [ LOGBUFF] Kbytes
Max # of Logical Logs [ LOGFILES]
Max # of Locks      [ LOCKS]
Max # of Buffers    [ BUFFERS]      Add SegSize (Kbytes) [SHMADD]
                                      Total Memory (Kbytes) [SHMTOTAL]

Shared memory size   [      546] Kbytes      Page Size [      2] Kbytes

Enter a unique value to be associated with this version of Informix database server
```

Figure 13-2
*ONCONFIG
Parameter for
Each of the
Shared-Memory
Entries*

For more information on the ONCONFIG parameters that are associated with the resident portion of shared memory, refer to [Figure 13-3 on page 13-11](#).

Setting Parameters for Resident Shared Memory with a Text Editor

You can use a text editor to set shared-memory configuration parameters at any time. Use the editor to 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 reinitialize shared memory.

Figure 13-3 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. For a description of the configuration parameters, refer to the *Administrator's Reference*.

Figure 13-3
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
SERVENUM	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

UNIX

Setting Parameters for Virtual Shared Memory with ON-Monitor

To set the configuration parameters for the virtual portion of shared memory with ON-Monitor, select the Shared-Memory option from the Parameters menu. [Figure 13-4](#) shows an example of a Shared-Memory screen. The shaded entries set configuration parameters for the virtual portion of shared memory.

Figure 13-4
Setting Virtual Shared-Memory Configuration Parameters

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.

                SHARED MEMORY PARAMETERS
Server Number          [ 2 ]      Server Name [lashley_o]      ]
Server Aliases [cole,davison,stackhouse]                        ]
Dbospace Temp [          ]                                           ]
Deadlock Timeout      [ 60] Secs  Number of Page Cleaners [ 1]
Forced Residency      [N]      Stack Size (Kbytes) [ 32]
Non Res. SegSize (Kbytes) [ 4000] Optical Cache Size (Kbytes) [ 0]

Physical Log Buffer Size [ 32] Kbytes  Dbospace Down Option [ 0]
Logical Log Buffer Size [ 32] Kbytes  Preserve Log for Log Backup [ N]
Max # of Logical Log [ 14]           Transaction Timeout [ 300]
Max # of Locks [ 2000]              Long TX HWM [ 50]
Max # of Buffers [ 80]              Long TX HWM Exclusive [ 60]
                                   Index Page Fill Factor [ 90]
                                   Add SegSize (Kbytes) [8192]
                                   Total Memory (Kbytes) [ 0]

Shared memory size [ 528] Kbytes      Page Size [ 2] Kbytes
Enter a unique value to be associated with this version of Informix database server
```

[Figure 13-5](#) shows only the Shared-Memory screen entries that affect the configuration of the virtual portion of shared memory. For each entry, it shows within brackets ([]) the name of the associated parameter in the ONCONFIG file.

Figure 13-5
ONCONFIG Parameters for Each of the Virtual Shared-Memory Entries

```
SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                SHARED MEMORY PARAMETERS

                                   Stack Size (Kbytes) [STACKSIZE]

Non Res. SegSize (Kbytes) [SHMVIRTSIZE]
```

For more information on the ONCONFIG parameters that affect the configuration of the virtual portion of shared memory, refer to [Figure 13-6](#).

Setting Parameters for Virtual Shared Memory with a Text Editor

You can use a text editor at any time to set the virtual shared-memory configuration parameters. Use the editor to locate the parameter in the file, enter the new value or values, and rewrite the file to disk.

[Figure 13-6](#) 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 13-6
Configuring the Virtual Portion of Shared Memory

ONCONFIG Parameter	Purpose
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 Performance Guide .
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 Performance Guide .
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory
STACKSIZE	Specifies the stack size for the database server user threads
SHMADD	Specifies the size of dynamically added shared-memory segments

(1 of 2)

ONCONFIG
Parameter

Purpose

SHMTOTAL	Specifies the total amount of memory to be used by the database server
STMT_CACHE	Specifies whether users can hold a parsed and optimized SQL statement in the SQL statement cache in memory. For more information about the SQL statement cache, refer to your Performance Guide .
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache

(2 of 2)

UNIX

Setting Parameters for Shared-Memory Performance Options with ON-Monitor

To set the configuration parameters for the shared-memory performance options with ON-Monitor, select the perFormance option from the Parameters menu. [Figure 13-7](#) shows the perFormance screen. The entries in the box set the configuration parameters for the shared-memory performance options.

Figure 13-7
*ON-Monitor
perFormance
Screen*

```
PERFORMANCE: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
PERFORMANCE TUNING PARAMETERS

Multiprocessor Machine      [  ]
  Num Procs to Affinity      [ 0]
  Proc num to start with     [ 0]

CPU VPs                      [ 1]
AIO VPs                      [  ]
Single CPU VP                [N]
Use OS Time                   [N]
Disable Priority Aging        [N]
Off-Line Recovery Threads    [10]
On-Line Recovery Threads     [ 1]
Num of LRUS queues           [ 8]

LRU Max Dirty                [ 60]
LRU Min Dirty                [ 50]
Checkpoint Interval          [300]
Num of Read Ahead Pages      [ 50]
Read Ahead Threshold         [ 40]

NETTYPE settings:
Protocol Threads Users VP-class
[soctcp] [ 1] [ 30] [NET]
[      ] [  ] [  ] [  ]
[      ] [  ] [  ] [  ]
[      ] [  ] [  ] [  ]
```


Figure 13-8 shows only the perFormance screen entries for setting the shared-memory performance options. For each entry, it shows within brackets ([]) the name of the associated parameter in the ONCONFIG file.

```
PERFORMANCE: Make desired changes and press ESC to record changes.  
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.  
PERFORMANCE TUNING PARAMETERS  
  
Num of LRUS queues      [LRUS]  
LRU Max Dirty           [LRU_MAX_DIRTY]  
LRU Min Dirty           [LRU_MIN_DIRTY]  
Checkpoint Interval     [CKPTINTVL]  
Num of Read Ahead Pages [RA_PAGES]  
Read Ahead Threshold    [RA_THRESHOLD]
```

Figure 13-8
*ONCONFIG
Parameters for the
Shared-Memory
Performance
Options*

For more information on the ONCONFIG parameters that set shared-memory performance options, refer to [Figure 13-9](#).

Setting Parameters for Shared-Memory Performance Options with a Text Editor

You can use a text editor to set ONCONFIG parameters at any time. To change one of the configuration parameters that set shared-memory performance options, use the text editor to locate the parameter in the file, enter the new value or values, and rewrite the file to disk. The changes that you make do not take effect until you reinitialize shared memory.

[Figure 13-9 on page 13-16](#) 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 13-9
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

Reinitializing Shared Memory

The database server reinitializes shared memory when you take the database server from off-line mode to quiescent mode or when you take it from off-line mode directly to on-line mode. To reinitialize shared memory, first bring the database server off-line. After the database server is off-line, bring it to quiescent mode or on-line mode to reinitialize shared memory. For information on how to take the database server from on-line mode to off-line, refer to [Chapter 8, “Managing Database Server Operating Modes.”](#)

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 on-line 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 12-18](#).

Turning Residency On or Off in On-Line Mode

To turn residency on or off while the database server is in on-line 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 NT, you must be a user in the **Informix Admin** group to turn residency on or off.

Turning Residency On or Off for the Next Time You Reinitialize Shared Memory

You can use a text editor to turn residency on or off for the next time that you reinitialize shared memory. 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 reinitialize shared memory.

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. [Figure 13-10](#) shows sample output.

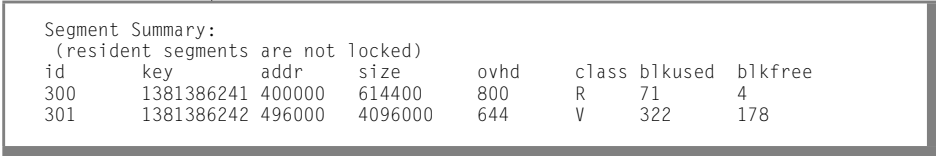


Figure 13-10
onstat -g seg Output

Monitoring the Shared-Memory Profile

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.

Using Command-Line Utilities

Execute **onstat -p** to display statistics on database server activity. [Figure 13-11](#) shows these statistics.

Figure 13-11
onstat -p Output

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				

The **onstat -p** output contains several fields that are not included in the information that the ON-Monitor Profile option displays. For a description of all the fields that **onstat** displays, see the utilities chapter in the *Administrator's Reference*.

Using ON-Monitor

Select Profile from the Status menu. The screen displays shared-memory statistics, as well as the current operating mode, the boot time, and the current time.

The field labels on the ON-Monitor Profile screen are easier to understand and are arranged in a slightly different order than the fields that appear if you execute **onstat -p**. However, all these statistics are included in the **onstat -p** output.

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**)

Figure 13-12 shows these fields.

```
Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
382      400      14438    97.35    381      568      3509    89.14
...
```

Figure 13-12
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} .

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
- Page numbers for all pages that remain in shared memory
- Address of the thread that currently holds the buffer
- Address of the first thread that is waiting for each buffer

Figure 13-13 shows an example of **onstat -B** output.

```
Buffers
address  userthread flgs  pagenum  memaddr  nslots  pgflgs  xflgs  owner  waitlist
849ae8   0          86   100955  84e000   1       b0     0     0     0      0
849b40   0          6   10095b  84e800   0       4     0     0     0      0
849b98   0          6   1009eb  84f000   0       4     0     0     0      0
849bf0   0          6   1008f5  84f800   2       70    0     0     0      0
...
84dea0   0          86   10093e  8b0800   8       1     0     0     0      0
84def8   0          6   10094b  8b1000   0       4     0     0     0      0
84df50   0          86   1009cd  8b1800   9       b0     0     0     0      0
0 modified, 200 total, 256 hash buckets, 2048 buffer size
```

Figure 13-13
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 13-14](#) 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
84a748   0          27  1012b0  860000   19     2001  80   8067c4  0
84add0   0          0   101752  869800   19     2001  80   807890  0
84b2a0   0          27  100c31  870800   19     2001  80   8067c4  0
84c798   0          27  10108e  88f000   19     2001  80   8067c4  0
84d818   0          27  101272  8a7000   19     2001  80   8067c4  0
154 modified, 200 total, 256 hash buckets, 2048 buffer size

```

Figure 13-14
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 13-21](#) for **onstat -p** (**bufreads**, **%cached**, **bufwrits** **%cached**), select the Profile option from the Status menu. [Figure 13-15](#) shows the output.

```
...
Disk Reads  Buff. Reads  %Cached  Disk Writes  Buff. Writes  %Cached
      177         330    46.36         4           0     0.00
...
```

Figure 13-15
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
bufwrits	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** command-line utility to obtain information on buffer-pool activity. 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 13-16](#) 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 13-16
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 12-50.) [Figure 13-17 on page 13-26](#) 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 13-17](#) shows an example of the **onstat -F** output.

Figure 13-17
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
```

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 13-18](#) shows an example of **onstat -R** output.

Figure 13-18
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%

```

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

Monitoring Latches

You can obtain statistics on latch use and information on specific latches.

The statistics include the number of requests for latches and the number of times that threads had to wait to obtain a latch. These statistics provide a measure of the system activity.

Information on specific latches includes a listing of all the latches that are held by a thread and any threads that are waiting for latches. This information allows you to locate any specific resource contentions that exist.

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about latches.

onstat -p

Execute **onstat -p** to obtain the values in the fields **lchreqs** and **lchwaits**. These fields store the number of requests for a latch and the number of times that a thread was required to wait for a shared-memory latch. A large number of latch waits typically results from a high volume of processing activity in which the database server is logging most of the transactions. (The administrator cannot configure or tune the number of latches; the database server sets this function internally.) [Figure 13-19](#) shows **onstat -p** output, including the **lchreqs** and **lchwaits** fields.

```
...
ixda-RA  idx-RA  da-RA  RA-pgsused  lchreqs  lchwaits
5        0      204    148        151762   12
```

Figure 13-19
onstat -p Output
Showing lchwaits
Field

onstat -s

Execute **onstat -s** to obtain general latch information. The output includes the **userthread** column, which lists the address of any user thread that is waiting for a latch. (See [Figure 13-20](#).) You can compare this address with the user addresses in the **onstat -u** output to obtain the user-process identification number.



Warning: *Never kill a database server process that is holding a latch. If you do, the database server immediately initiates an abort.*

Latches with lock		or userthread set		
name	address	lock	wait	userthread
LRU1	402e90	0	0	6b29d8
bf[34]	4467c0	0	0	6b29d8

Figure 13-20
onstat -s Output

UNIX

Using ON-Monitor

To access the **latch Waits** field, mentioned on [page 13-28](#) for **onstat -p**, select the Profile option from the Status menu.

Using SMI Tables

Query the **sysprofile** table to obtain the number of requests for a latch and the number of times a thread had to wait for a latch. The following rows are relevant.

Row	Description
latchreqs	Number of requests for a latch
latchwts	Number of times that a thread had to wait for a latch

Data Storage

In This Chapter	14-3
Overview of Data Storage	14-3
Physical Units of Storage	14-5
Chunks	14-5
Uses of Chunks	14-5
Chunk Size, Number, and Names	14-6
Disk Allocation for Chunks	14-6
Unbuffered or Buffered Disk Access on Windows NT	14-6
Unbuffered or Buffered Disk Access on UNIX	14-7
Offsets	14-10
Pages	14-11
Blobpages.	14-12
Sbpages	14-14
Extents.	14-16
Disabling I/O Errors	14-18
Logical Units of Storage	14-19
Dbspaces	14-19
Control of Where Data Is Stored	14-20
Root Dbspace	14-22
Temporary Dbspaces	14-22
Advantages of Using Temporary Dbspaces	14-23
Blobspaces	14-24
Sbspaces	14-24
Advantages of Using Sbspaces	14-25
Metadata and User Data	14-26
Control of Where Data Is Stored	14-27
Storage Characteristics of Sbspaces	14-29
Other Attributes of Sbspaces	14-30
More Information About Sbspaces	14-34

Extspaces	14-35
Databases	14-36
Tables	14-37
Temporary Tables	14-38
Storage of Temporary Tables	14-40
Tblspaces	14-42
Extent Interleaving	14-44
Table Fragmentation and Data Storage	14-45
Amount of Disk Space Needed to Store Data	14-45
Size of the Root Dbspace.	14-45
Physical and Logical Logs	14-46
Temporary Tables.	14-46
Critical Data	14-47
ON-Archive Catalog Data	14-47
Control Information	14-47
Amount of Space That Databases Require.	14-48
Disk-Layout Guidelines	14-48
Dbspace and Chunk Guidelines	14-49
Strive to Associate Partitions with Chunks	14-49
Mirror Critical Data Dbspaces	14-49
Spread Temporary Storage Space Across Multiple Disks	14-50
Move the Logical and Physical Logs from the Root Dbspace	14-50
Consider Account Backup-and-Restore Performance	14-51
Table-Location Guidelines	14-52
Isolate High-Use Tables.	14-52
Consider Mirroring	14-54
Group Tables with Backup and Restore in Mind	14-54
Place High-Use Tables on Middle Partition of Disk	14-55
Optimize Table-Extent Sizes	14-55
Sample Disk Layouts	14-56
Sample Layout When Performance Is Highest Priority	14-57
Sample Layout When Availability Is Highest Priority	14-59
Logical-Volume Manager	14-61

In This Chapter

This chapter defines terms and explains the concepts that you must understand to perform the tasks described in [Chapter 15, “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 [“Documentation Notes, Release Notes, Machine Notes” on page 14](#) of the Introduction.

Overview of Data Storage

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

- Chunk
- Page
- Extent
- Blobpage
- Sbpage

Overlying the physical units of storage space, the database server supports the following logical units associated with database management:

- Dbspace
- Blobspace
- Extspace
- Sbspace
- Database
- Table
- Tblspace

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

- Logical log
- Physical log
- Reserved pages

Because these additional disk-space structures are not permanent storage units, they are not described in this chapter. For information about the logical log, see [Chapter 19, “Logical Log.”](#) For information about the physical log, see [Chapter 21, “Physical Logging.”](#) For information about reserved pages, see the disk structures and storage chapter in the *Administrator’s Reference*.

The following sections describe the various data-storage units that the database server supports and the relationships between those units.

Physical 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 following sections describe the physical units of storage in more detail.

Chunks

A *chunk* is the largest unit of physical disk dedicated to database server data storage. Chunks provide administrators with a conveniently large unit for allocating disk space.

Some operating systems use the concept of a logical volume, and others use a logical unit. Each of these terms represents the smallest unit of physical disk that you can assign. A database server chunk is the same as a logical volume or a logical unit.

Uses of Chunks

The database server administrator assigns one or more chunks to the following logical storage spaces that the database server supports:

- Dbspace
- Blobspace
- Sbspace

The database server administrator typically adds a chunk to one of these storage spaces when that storage space approaches full capacity. For more information on these logical storage spaces, refer to [“Logical Units of Storage” on page 14-19](#).

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 on-line automatically. For more information on mirroring, see [Chapter 24, “Mirroring.”](#)

Chunk Size, Number, and Names

Chunk names follow the same rules as dbspace names. For more details, see [“Naming Chunks and Storage Spaces” on page 15-18](#).

For information on the maximum size and number of chunks that you can allocate on the database server system, see [“Limiting Chunk Size and Number” on page 15-19](#).

Disk Allocation for Chunks

The database server can use regular operating-system files to store data. On operating systems that support raw disks, the database server can also use raw disk space to store data. On UNIX, Informix recommends that you use raw disks to store data whenever performance or data consistency is important.

WIN NT

Unbuffered or Buffered Disk Access on Windows NT

On Windows NT, the database server uses unbuffered I/O to access both *raw disk space* and NTFS files. The Windows NT file system manager adds additional overhead to disk I/O, so using raw disk space provides some performance advantages. However, because NTFS files are a more standard method of storing data, Informix recommends you use NTFS files instead of raw disk space unless your database server performs a large amount of disk access that requires the best performance.

NTFS Files

If you use files for disk space on Windows NT, the files must be NTFS files.

Raw Disk Space on Windows NT

In the Windows NT environment, *raw disk space* can be either a physical drive without a drive letter or a disk partition that has been assigned a drive letter using the Windows NT 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.

UNIX

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.

Unbuffered disk access can be through a raw disk device, or character-special files. As a general guideline, you experience better performance and increased reliability when you use unbuffered file access.

On UNIX, the raw disk interface that character-special files provide yields significant performance advantages. I/O to raw disk bypasses the buffering operations that the operating system performs on regular (cooked) files.

Raw Disk Space on UNIX

UNIX uses the concept of a *device* to describe peripherals such as magnetic disks and tapes, terminals, and communication lines. One type of device is a *block device*, such as a hard disk or a tape. A block device can be configured with an interface that provides buffering or with a raw interface that leaves the buffering to the application. When you configure a block device with a raw interface, the device is called a *raw device*, and the storage space that the device provides is called *raw disk space*. Space in a chunk of raw disk space is physically contiguous.

A raw interface is also referred to as a *character-special device*. 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/rsd0f** is the character-special device that corresponds to the **/dev/sd0f** block-special device.

Cooked Files

A cooked file is a regular file that the operating system manages. Although the database server manages the contents of cooked files, the operating system manages all I/O to cooked files. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

Even though a cooked file is a regular file, the database server manages the internal arrangement of data within the file. Never edit the contents of a cooked file that the database server manages. To do so puts the integrity of your data at risk.

Data Management with Cooked Files Versus Raw Disk Devices

When the operating system reads from a cooked file, it reads the data from disk to an internal buffer pool. Later, a second copy operation copies it from the operating system to the location requested by the application. Therefore, when two users both read the same file, the data is read from disk only once but copied from the operating-system buffer twice.

By contrast, when the operating system reads data from an unbuffered file or a raw disk device, it bypasses the operating-system buffer pool and copies the data directly to the location requested by the application. The database server requests that the data be placed in shared memory, making it immediately available to all database server virtual processors and running threads with no further copying.

Unbuffered Disk Access

A raw device or unbuffered file can directly transfer data between shared memory and the disk with direct memory access (DMA), which results in better performance by orders of magnitude.

When you use a raw device or unbuffered file to store your data, the database server guarantees that committed data is stored on disk. (The next section explains why no such guarantee can be made when you use cooked files to store your data.)

When you decide to allocate raw disk space to store your data, you must take the following steps:

1. Create and install a raw device.
2. Change the ownership and permissions of the device.

For more information on these steps, see [“Allocating Raw Disk Space on UNIX” on page 15-10](#).

Use of Cooked Files

You can more easily allocate cooked files than raw disk space. To allocate raw space, you must have a disk partition available that is dedicated to raw space. To allocate a cooked file, you need only create the file on any existing partition. However, you sacrifice reliability and might experience diminished performance when you store the database server data in cooked files.

The buffering mechanism that most operating systems provide can become a performance bottleneck. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system located near the center cylinders of the disk device or in a file system with minimal activity.

In a learning environment, where reliability and performance are not critical, cooked files can be convenient.

When performance is not a consideration, you can also use cooked files for static data (which seldom or never changes). Such data is less vulnerable to the problems associated with UNIX buffering in the event of a system failure.

When a chunk consists of cooked disk space, the name of the chunk is the complete pathname of the file. Because the chunk of cooked disk space is an operating-system file, space in the chunk might not be physically contiguous.



Warning: *Cooked files are less reliable than raw disk space because the operating system manages I/O for a cooked file. A write to a cooked file can result in data being written to a memory buffer in the operating-system file manager instead of being written immediately to disk. As a consequence, the database server cannot guarantee that the committed data actually reaches the disk. Database server recovery depends on the guarantee that data written to disk is actually on disk. In the event of system failure, if the data is not present on disk, the database server automatic-recovery mechanism might not be able to execute properly. The end result would be inconsistent data.*

When you decide to allocate cooked space to store your data, you must take the following steps:

1. Create a cooked file.
2. Change the ownership and permissions.

These steps are described in detail in [“Allocating a File for Disk Space on UNIX” on page 15-9](#).

Offsets

The system administrator might divide a physical disk into *partitions*, which are different parts of a disk that have separate pathnames. Although Informix recommends 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 [“Strive to Associate Partitions with Chunks” on page 14-49](#).

An offset allows you to indicate the number of kilobytes into a device or cooked file to reach a given chunk. 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`, `blob space`, or `sbspace`.

The maximum offset that you can specify is 2 gigabytes. The maximum chunk size is also 2 gigabytes.

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 15-10](#) 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 14-1](#) illustrates the concept of a page, represented by a darkened sector of a disk platter.

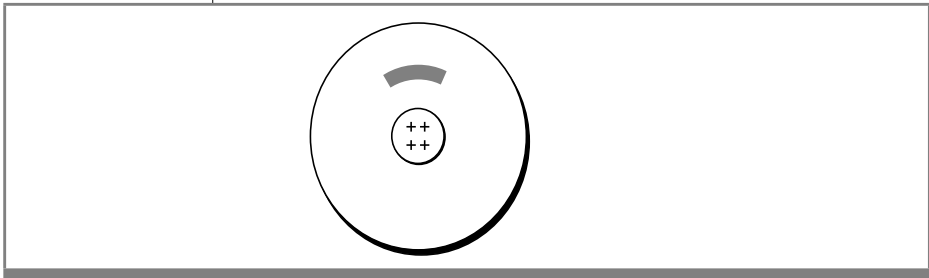


Figure 14-1
A Page on Disk

On most UNIX platforms, the page size is 2 kilobytes. On Windows NT, 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 14-2](#) 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*.

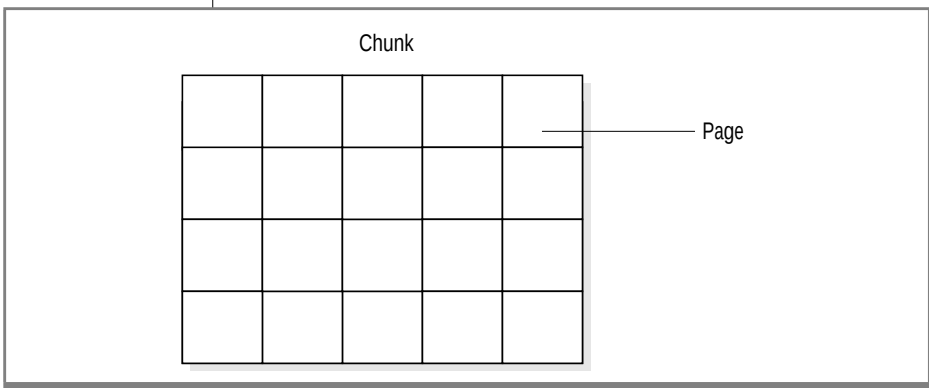


Figure 14-2
A Chunk, Logically Separated into a Series of Pages

Blobpages

A blobpage is the unit of disk-space allocation that the database server uses to store simple large objects (TEXT or BYTE data) within a blobpage. For a description of blobspaces, refer to “[Blobspaces](#)” on page 14-24.

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 14-3](#) illustrates the concept of a blobpage, represented as a multiple (three) of a data page.

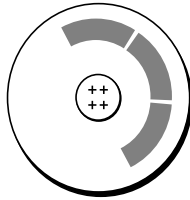


Figure 14-3
A Blobpage on Disk

For information about how Dynamic Server structures data stored in a blobpage, see structure of a 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 14-4](#) illustrates. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

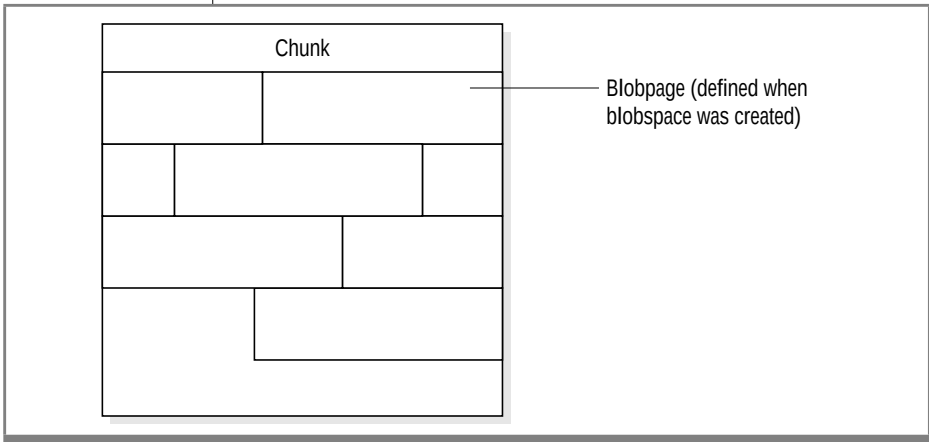


Figure 14-4
A Chunk in a
Blobspace, Logically
Separated into a
Series of Blobpages

Instead of storing simple-large-object data in a blobspace, you can choose to store it in a dbspace. However, for a simple large object larger than two pages, performance improves when you store it in a blobspace. Simple large objects stored in a dbspace can share a page, but simple large objects stored in a blobspace do not share pages.

For information on how to determine the size of a blobpage, refer to [“Determining Blobpage Size”](#) on page 15-21.

Sbpages

An sbpage is the type of page that the database server uses to store smart large objects within a sbspace. For a description of sbspaces, refer to [“Sbspaces” on page 14-24](#). 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 NT.

The unit of allocation in an sbspace is an extent whereas the unit of allocation in a blobspace is a blobpage. 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.

If you know the size of the smart large object, Informix recommends that you specify this size in one of the following ways:

- The DataBlade API **mi_lo_specset_estbytes()** function
For more information on the DataBlade API functions for smart large objects, refer to the [DataBlade API Programmer's Manual](#).
- The ESQL/C **ifx_lo_specset_estbytes** function.
For more information on the ESQL/C functions for smart large objects, refer to the [Informix ESQL/C Programmer's Manual](#).

These functions are the best way to set the extent size because the database server allocates the entire smart large object as one extent (if an extent of that size is available in the chunk).



Important: For most applications, Informix recommends that you use the values that the database server calculates for the extent size. However, programmers can specify the DataBlade API **mi_lo_specset_estbytes** function or the corresponding ESQL/C **ifx_lo_specset_estbytes** function to indicate to the database server the expected size of the smart large object.

Just as with pages in a chunk, a certain number of smart large object extents compose a chunk in an sbpace, as [Figure 14-5](#) illustrates. An extent is always entirely contained in a chunk and cannot cross chunk boundaries.

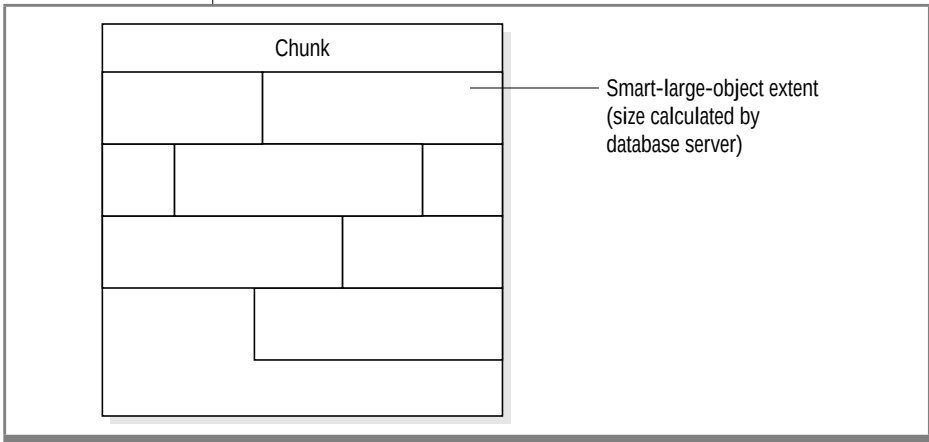


Figure 14-5
*A Chunk in an
Sbpace, Logically
Separated into a
Series of Extents*

Smart large objects cannot be stored in a dbspace. For information about how Dynamic Server structures data stored in an sbpace, see sbpace structure in the disk structures and storage chapter of the *Administrator's Reference*. For more information on sbspaces, refer to [“Sbspaces” on page 14-24](#).

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 14-6](#) illustrates the concept of an extent.

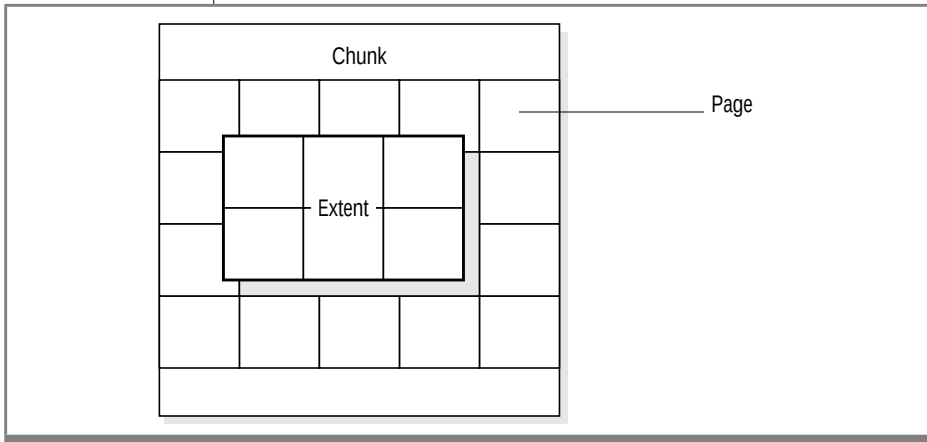


Figure 14-6
An Extent That
Consists of Six
Contiguous Pages
on a Raw Disk
Device

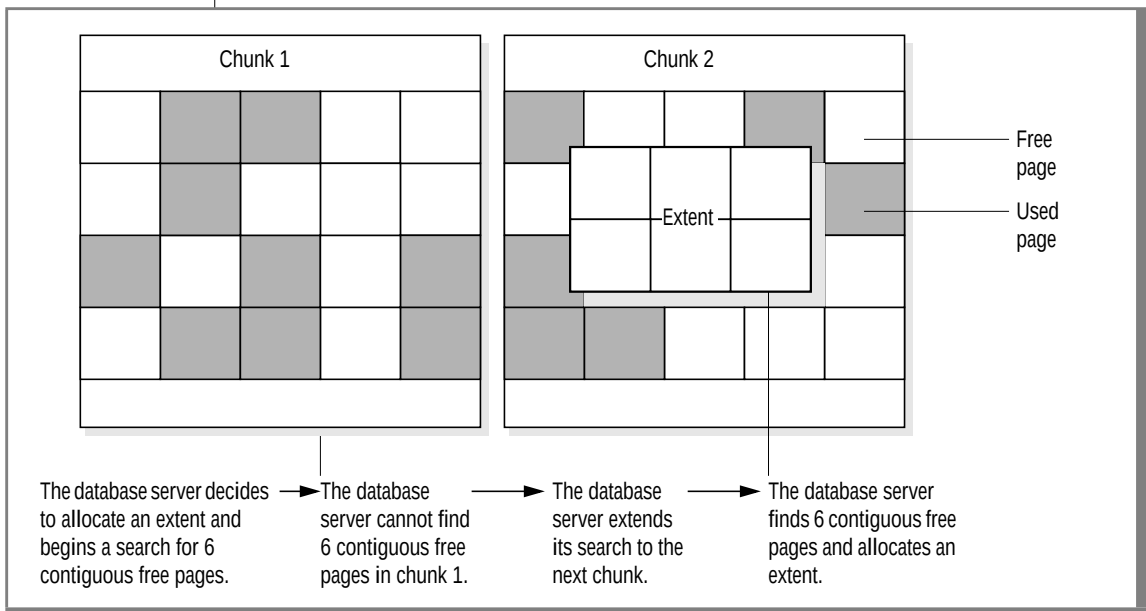
An extent consists of a collection of contiguous pages that store data for a given table. (See [“Tables” on page 14-37](#).) Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of kilobytes allocated to the table when it is first created. The *next-extent* size is the number of kilobytes allocated to the table when the initial extent (and any subsequent extents) becomes full. To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the [Informix Guide to SQL: Syntax](#).

When you create a table with a column for CLOB or BLOB data types, you also define extents for an sbpace. For more information, refer to [“Storage Characteristics of Sbspaces” on page 14-29](#).

Figure 14-7 illustrates the following key concepts concerning extent allocation:

- 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 (six pages in Figure 14-7), it searches the next chunk in the dbspace for contiguous space.

Figure 14-7
Process of Extent Allocation



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 off-line.

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 off-line and then on-line 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.

Logical Units of Storage

The logical units of database server storage fall into the following categories:

- Units of logical storage that are dictated by relational database design:
 - Databases
 - Tables
- Units of logical storage that function as accounting entities:
 - Dbspaces
 - Blobspaces
 - Extspaces
 - Sbspaces
 - Tbspaces

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 14-37.](#))

The following sections describe these logical storage units.

Dbspaces

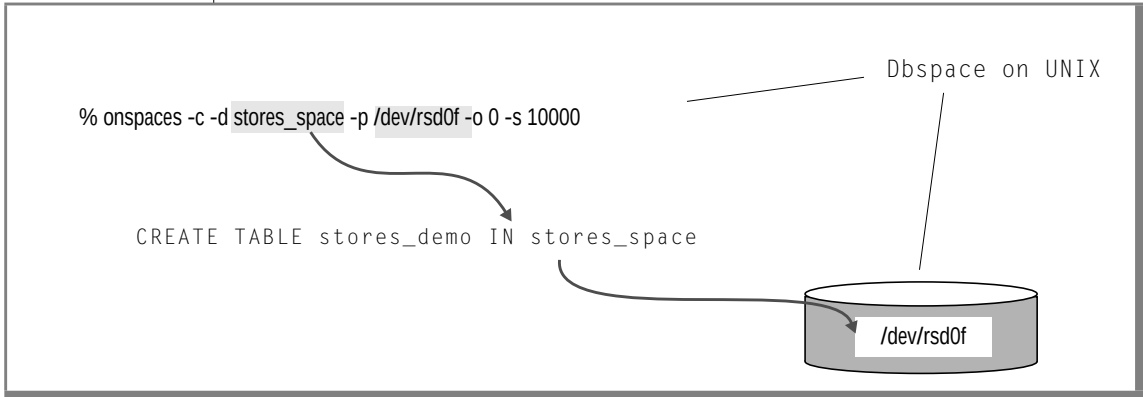
A key responsibility of the database server administrator is to control where the database server stores data. By storing high-use access tables or critical *dbspaces* (root *dbspace*, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on 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.

These strategies require the ability to control the location of data. The logical storage unit that provides this ability is the *dbspace*. The *dbspace* provides the critical link between the logical and physical units of storage. It allows you to associate physical units (such as chunks) with logical units (such as tables).

Control of Where Data Is Stored

As [Figure 14-8](#) 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 14-37.](#))

Figure 14-8
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 DbSPACE” on page 15-13.](#)

A dbspace includes one or more chunks, as [Figure 14-9](#) 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 the Database Server for Disabling I/O Errors” on page 15-44.](#)) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

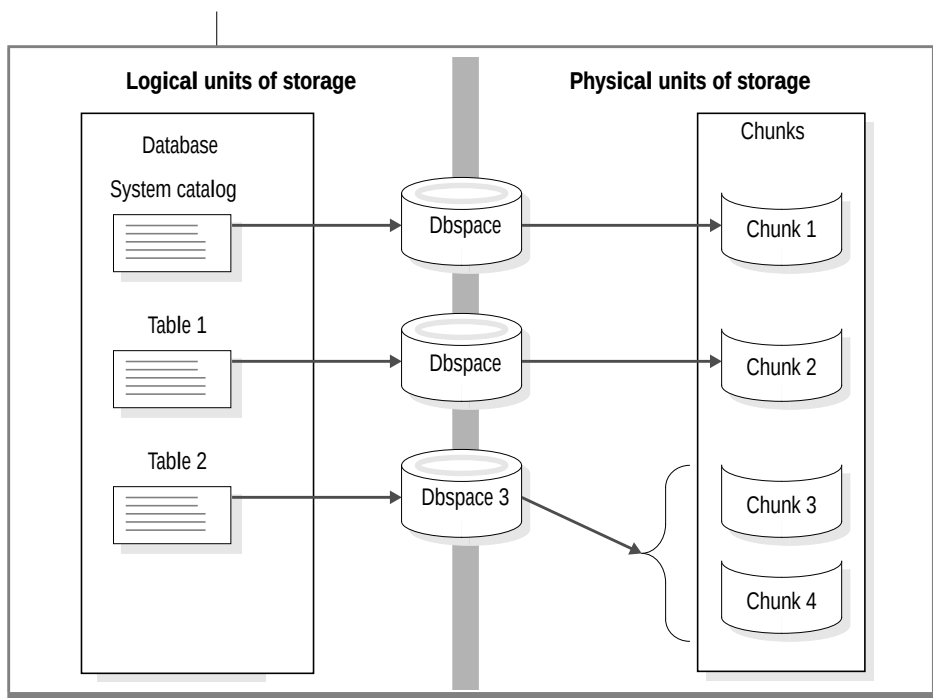


Figure 14-9
Dbspaces That Link
Logical and Physical
Units of Storage

The database server uses the dbspace to store databases and tables. (See [“Tables” on page 14-37.](#))

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 15-46.](#)

You can use ON-Monitor (on UNIX) or **onspaces** to perform any of the following tasks related to dbspace management:

- Creating a dbspace (See [page 15-13.](#))
- Adding a chunk to a dbspace (See [page 15-17.](#))
- Dropping a chunk from a dbspace (See [page 15-30.](#))
- Dropping a dbspace or blobspace (See [page 15-31.](#))

Root Dspace

The root dspace is the initial dspace that the database server creates. The root dspace 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 14-37](#) and the disk structures and storage chapter in the *Administrator’s Reference*.) The initial chunk of the root dspace and its mirror are the only chunks created during disk-space initialization. You can add other chunks to the root dspace after disk-space initialization.

The following disk-configuration parameters in the ONCONFIG configuration file refer to the first (initial) chunk of the root dspace:

- ROOTPATH
- ROOTOFFSET
- ROOTNAME
- MIRRORPATH
- MIRROROFFSET

The root dspace is also the default dspace location for any database created with the CREATE DATABASE statement.

The root dspace is the default location for all temporary tables created implicitly by the database server to perform requested data management.

[“Size of the Root Dspace” on page 14-45](#) explains how much space to allocate for the root dspace. You can also add extra chunks to the root dspace after you initialize database server disk space.

Temporary Dbspaces

A temporary dspace is a dspace reserved for the exclusive use of temporary tables. (See [“Temporary Tables” on page 14-38](#).)

The database server never drops a temporary dspace unless it is explicitly directed to do so. A temporary dspace is temporary only in the sense that the database server does not preserve any of the dspace contents when the database server shuts down abnormally. Temporary dbspaces are designed exclusively for the storage of temporary tables.

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. Backup utilities do not include temporary dbspaces as part of a full-system dbspace backup. You cannot mirror a temporary dbspace.

For detailed instructions on how to create a temporary dbspace, see [“Creating a Temporary Dbpace” on page 15-15](#).



Important: *When the database server is running as a secondary database server in a high-availability data-replication pair, it requires a temporary dbspace to store any internal temporary tables generated by read-only queries.*

Advantages of Using Temporary Dbspaces

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 suppresses all logical logging for implicit temporary tables and explicit temporary tables created with the WITH NO LOG options that reside in a temporary dbspace. 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.

The database server does not perform any physical logging in temporary dbspaces. This practice helps performance in two ways. First, physical logging itself generates I/O. Reducing I/O always improves performance. Second, whenever the physical log becomes 75 percent full, a checkpoint occurs. Checkpoints require a brief period of inactivity to complete, which can have a negative impact on performance. When temporary tables reside in temporary dbspaces, the database server does not perform physical logging for operations on the temporary tables, thus requiring fewer checkpoints.

Using temporary dbspaces to store temporary tables also reduces the size of your dbspace backup because the database server does not backup temporary dbspaces.

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 14-37](#)) 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 15-56](#). For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see [Chapter 15, “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 [Informix Guide to SQL: Reference](#).

Advantages of Using Sbspaces

Sbspaces have the following advantages over blobspaces:

- Read, write, and seek properties similar to a standard UNIX file
Unlike the large objects stored in blobspaces, the smart large objects stored in sbspaces have read, write, and seek properties similar to a standard UNIX file. Programmers can use functions similar to UNIX and Windows NT 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.
- 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.
- 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](#).
- Data for smart large objects not required to be transferred in its entirety
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 [DataBlade API Programmer's Manual](#). For more information on the ESQL/C functions, refer to the [Informix ESQL/C Programmer's Manual](#).

Metadata and User Data

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 two 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.

For information on correctly allocating metadata and user data for sbspaces, see [“Sizing Sbspace Metadata” on page 15-25](#).

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. For instructions on how to create a sbspace, add chunks to a sbspace, or drop a chunk from a sbspace, see [Chapter 15, “Managing Disk Space.”](#)

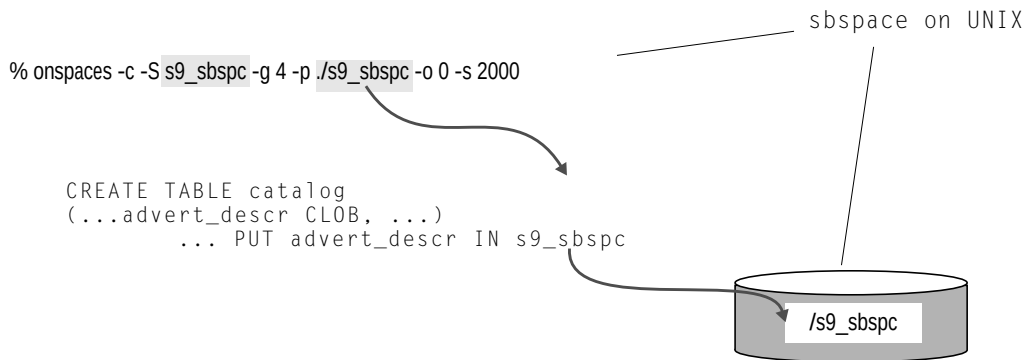


Important: *Sbspace metadata is always logged, regardless of the logging setting of the database.*

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 14-10](#) 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 14-10
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 Blobspace”](#) on page 15-24. For more information on how to specify smart-large-object characteristics in the PUT clause, refer to the CREATE TABLE statement in the [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 14-11](#) 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 sbspaces for fullness and to anticipate the need to allocate more chunks to a sbspaces. For more information on monitoring sbspaces, refer to your [Performance Guide](#).

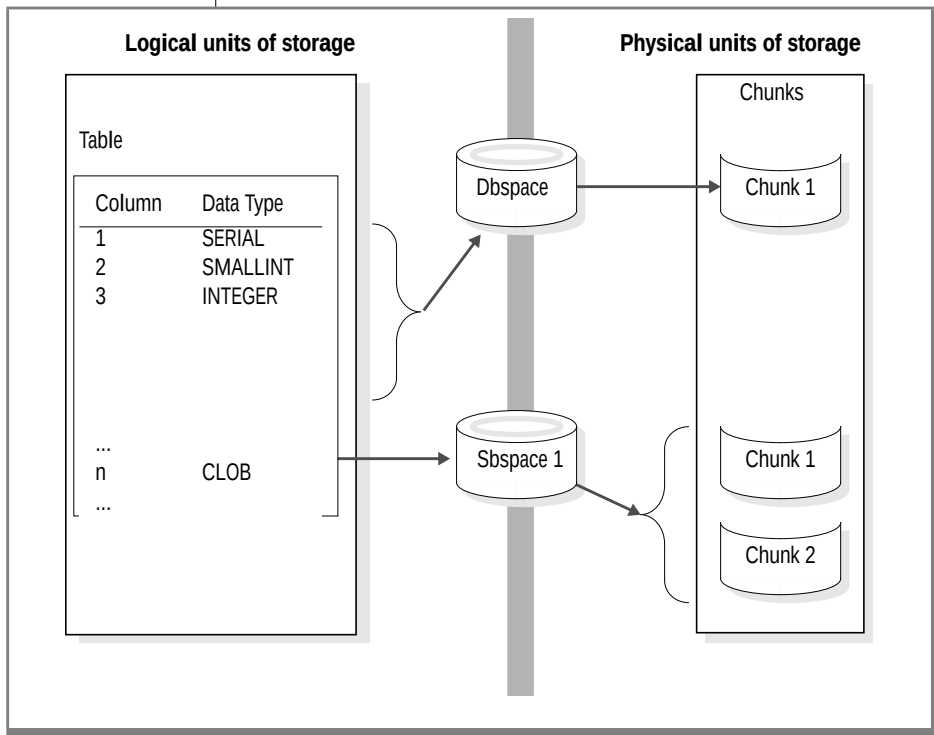


Figure 14-11
*Sbspaces That Link
Logical and Physical
Units of Storage*

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 sbpace to speed recover in event of a media failure. For more information, refer to [“Mirroring” on page 24-4](#).

You can use **onspaces** to perform any of the following tasks related to sbspac management:

- Creating an sbspac (Refer to [page 15-24](#).)
- Adding a chunk to an sbspac (Refer to [page 15-27](#).)
- Dropping a chunk from an sbspac (Refer to [page 15-30](#).)
- Dropping an sbspac (Refer to [page 15-31](#).)

Storage Characteristics of Sbspaces

When you create an sbspac, you can use the system defaults or change the values for the following storage characteristics:

- Extent sizes
- Average smart-large-object size

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 sbspac with **onspaces -c**. Later on, you can change these sbspac 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 sbspac consists of a collection of contiguous pages that store smart-large-object data.

The unit of allocation in an sbspac 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, Informix recommends that you use the values that the database server calculates for the extent size. However, if you know the size of the smart large object, Informix recommends that programmers specify this size in the DataBlade API **mi_lo_specset_estbytes** function or the ESQL/C **ifx_lo_specset_estbytes** function instead of in the **onspaces** utility or the **CREATE TABLE** or the **ALTER TABLE** statement. This **mi_lo_specset_estbytes()** function (and the corresponding ESQL/C **ifx_lo_specset_estbytes()** function) is the best way to set the extent size because the database server allocates the entire smart large object as one extent.

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.

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 15-25](#).

Other Attributes of Sbspaces

When you create an sbspace, you can specify the following additional attributes:

- Buffering mode
- Last-access time
- Lock mode
- Logging

As the database server administrator, you can use the system default values for these attributes, or you can override them for specific sbspaces, columns in a table, or smart large objects. The database server uses the *storage-characteristics hierarchy*, which [Figure 14-12](#) shows, to obtain the storage characteristics for a smart large object.

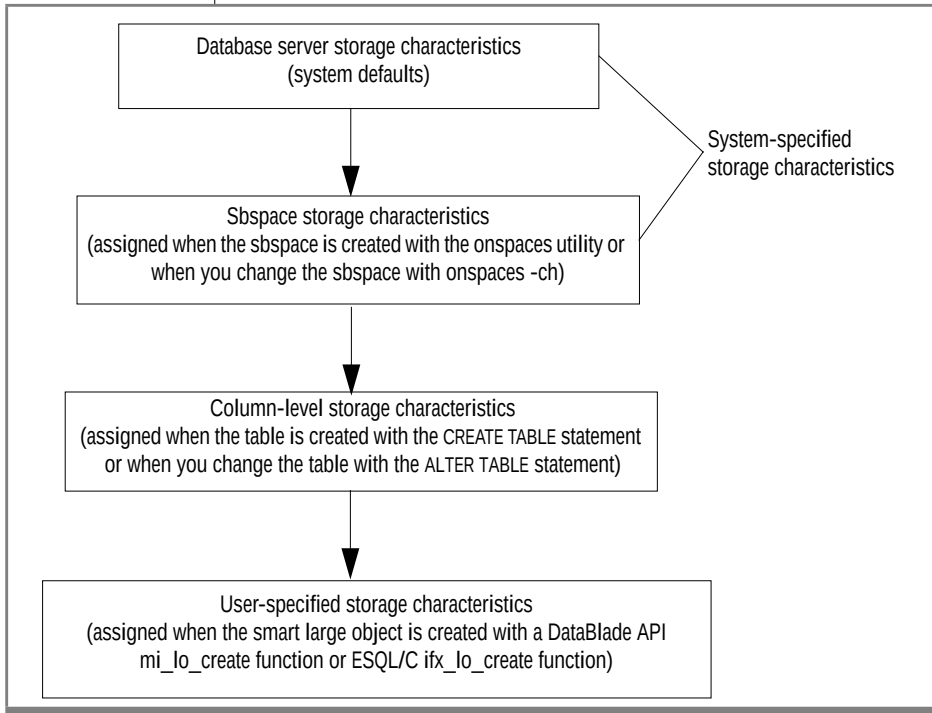


Figure 14-12
Storage-Characteristics Hierarchy

[Figure 14-12](#) shows that you can override the system default in the following ways:

- You override the system default for a specific sbspace when you them in the **-Df** tags when you create the sbspace with **onspaces -c**. You can later change these sbspace attributes for the sbspace with the **onspaces -ch** option. For more information on valid ranges for the different **-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 [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.

For more information on the DataBlade API functions for smart large objects, refer to the [DataBlade API Programmer's Manual](#). For more information on the ESQL/C functions for smart large objects, refer to the [Informix ESQL/C Programmer's Manual](#).

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 ACESSTIME tag of the **onspaces -c -Df** option. The default is “ACESSTIME=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 [DataBlade API Programmer's Manual](#) and *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 "Sbspace Logging" on page 19-32.



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 [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 [DataBlade API Programmer's Manual](#). For more information on the ESQL/C functions for smart large objects, refer to the [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 12-58.

More Information About Sbspaces

Figure 14-13 lists sources of information about various tasks related to using and managing sbspaces.

Figure 14-13
Finding Information for Sbspace Tasks

Task	Reference
Creating a table with CLOB or BLOB data types	Informix Guide to SQL: Syntax
Defining or altering storage characteristics for a smart large object.	“Altering Storage Characteristics of Smart Large Objects” on page 15-28 , PUT clause of CREATE TABLE or ALTER TABLE statement in Informix Guide to SQL: Syntax , DataBlade API functions in DataBlade API Programmer’s Manual , ESQL / C functions in Informix ESQL/C Programmer’s Manual
Accessing smart large objects in an application	DataBlade API Programmer’s Manual , Informix ESQL/C Programmer’s Manual
Setting memory configuration parameters for smart large objects	Chapter 13, “Managing Shared Memory”
Monitoring and tuning shared memory for sbspaces	Chapter on configuration effects on memory utilization in Performance Guide
Allocating space for an sbspace	“Creating an Sbspace” on page 15-24
Adding a chunk to an sbspace	“Adding a Chunk to an Sbspace” on page 15-27
Calculating the metadata area	Chapter on table performance in your Performance Guide
Monitoring and tuning space within an sbspace	Chapter on table performance in your Performance Guide
Understanding smart-large-object locking	Chapter on locking in your Performance Guide
Backing up an sbspace	“Backup of Sbspace Data and Log Records” on page 19-33

Task	Reference
Logging considerations for an sbpace	“Logging of Smart Large Objects” on page 17-9, “Sbpace Logging” on page 19-32
Specifying I/O characteristics for an sbpace	onspaces option in “Storage Characteristics of Sbspaces” on page 14-29 , DataBlade API functions in DataBlade API Programmer’s Manual , ESQL/C functions in Informix ESQL/C Programmer’s Manual
Monitoring and tuning I/O utilization for an sbpace	Chapter on configuration effects on I/O utilization in your Performance Guide
Understanding an sbpace structure	Chapter on disk structures in the Administrator’s Reference
Using onspaces for sbspaces	Chapter on utilities in the Administrator’s Reference
Checking consistency of an sbpace	“Validating Metadata” on page 29-5

(2 of 2)

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 [Virtual-Table Interface Programmer's Manual](#). For more information on creating functions and primary access methods, see the [Informix Guide to SQL: Syntax](#).

Databases

A database is a logical storage unit that contains tables and indexes. (See “Tables” on page 14-37.) 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 14-14 shows the tables contained in the **stores_demo** database.

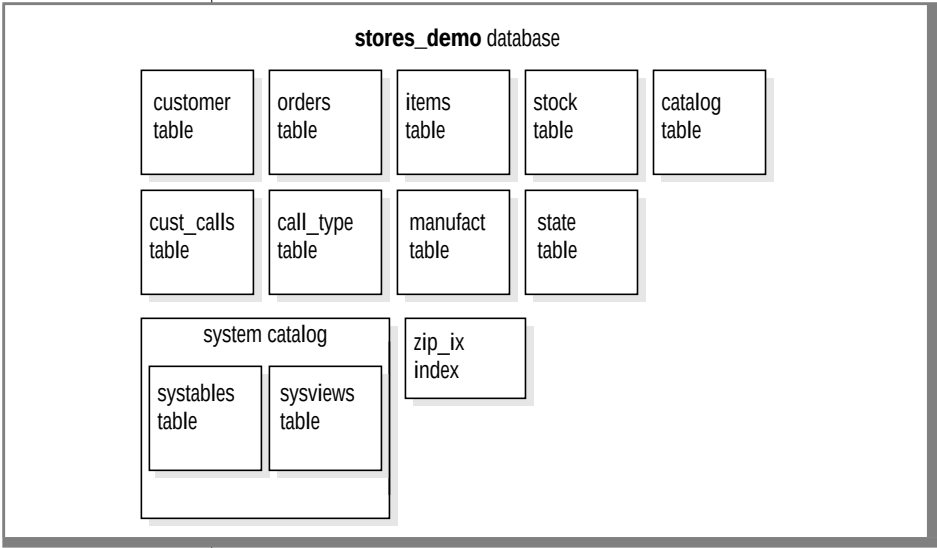


Figure 14-14
The stores_demo Database

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 15-43](#).

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 14-16](#).) 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.

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. For more information about distribution schemes, see the [Informix Guide to Database Design and Implementation](#).

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.

A table, composed of extents, can span multiple chunks, as [Figure 14-15](#) shows.

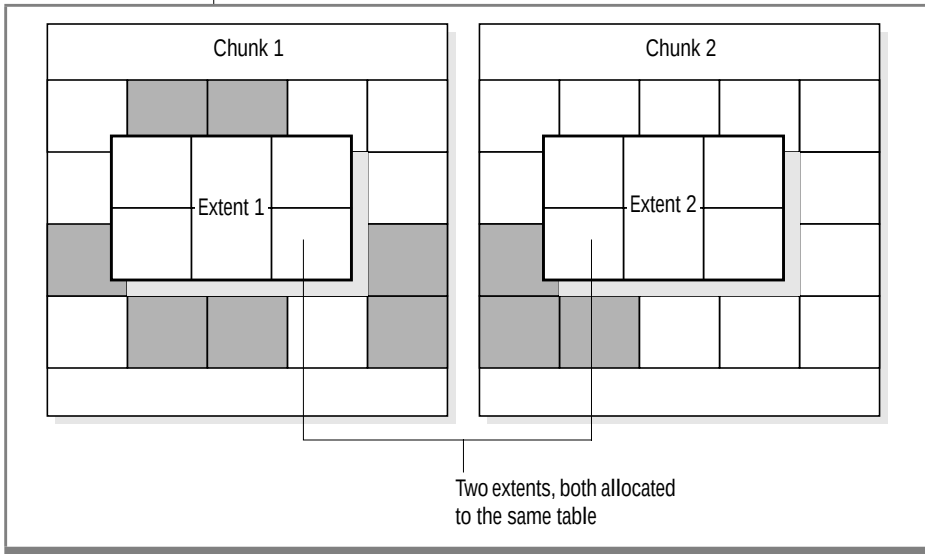


Figure 14-15
*Table That Spans
More than One
Chunk*

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 14-48](#) and your [Performance Guide](#).

Temporary Tables

The two types of temporary tables are *explicit* temporary tables and *implicit* temporary tables. You can create temporary tables in a standard dbspace or temporary dbspace.

An *explicit* temporary table is a temporary table that you create with one of the following SQL statements:

- TEMP TABLE option of the CREATE TABLE statement
- INTO TEMP clause of the SELECT statement

```
SELECT * FROM customer INTO TEMP temp_table
```

When an application creates an explicit temporary table, it exists until the application takes one of the following actions:

- The application terminates.
- The application closes the database in which the table was created and opens a database in a different database server.
- The application closes the database in which the table was created. In this case, the database server drops the table only when the database uses transaction logging.

When any of these three events occurs, the database server deletes the temporary table.

An *implicit* temporary table is a temporary table or file that the database server creates as part of processing.

The following statements might require temporary disk space:

- 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
- DECLARE statements that use the SCROLL CURSOR option

The database server deletes an implicit temporary table when the processing that initiated the creation of the table is complete.

If the database server shuts down without adequate time to clean up temporary tables, it performs temporary table cleanup as part of the next initialization. (To request shared-memory initialization without temporary table cleanup, execute **oninit** with the **-p** option.)

.

Storage of Temporary Tables

Adequate temporary space in the appropriate dbspaces to store temporary tables and files is critical to the overall performance of your database server. The output file of the SQL statement SET EXPLAIN ON lists temporary-file requirements.

The dbspace in which the database server stores temporary tables depends on whether the table is an explicit or implicit table. The following sections examine both cases in detail.

Explicit Temporary Tables

When you create an explicit temporary table using the IN *dbspace* option of CREATE TEMP TABLE, the database server stores the temporary table in that dbspace.

When you do not use the IN *dbspace* option of CREATE TEMP TABLE, or when you create the explicit table with SELECT... INTO TEMP, the database server checks the **DBSPACETEMP** environment variable and the DBSPACETEMP configuration parameter. (The environment variable supersedes the configuration parameter.) When **DBSPACETEMP** is set, the database server stores the explicit temporary table in one of the dbspaces specified in the list.

The database server keeps track of the last dbspace in the list that it used to store a temporary table. When the database server receives another request for temporary storage space, it uses the next dbspace in the list. In this way, the database server spreads I/O evenly across the temporary storage space that you specify in **DBSPACETEMP**.



Important: In the case of a database with logging, you must include the WITH NO LOG clause in the SELECT... INTO TEMP statement to place the explicit temporary tables in the dbspaces listed in the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable.

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 (nontemporary) 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.

Implicit Temporary Tables

The database server stores implicit temporary tables in one of the dbspaces that you specify in the **DBSPACETEMP** environment variable or the **DBSPACETEMP** configuration parameter. The environment variable supersedes the configuration parameter.

When the **DBSPACETEMP** environment variable and the **DBSPACETEMP** configuration parameter are not set, Dynamic Server stores the temporary table in the root dbspace. When the database server creates implicit temporary tables in the process of sorting, it checks the **PSORT_DBTEMP** environment variable in addition to checking the **DBSPACETEMP** environment variable and the **DBSPACETEMP** configuration parameter. For more information, see your [Performance Guide](#).

For information on how to create temporary dbspaces, refer to [“Creating a Temporary Dbspace” on page 15-15](#).

Tblspaces

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.

The table *tblspace* contains the following types of pages:

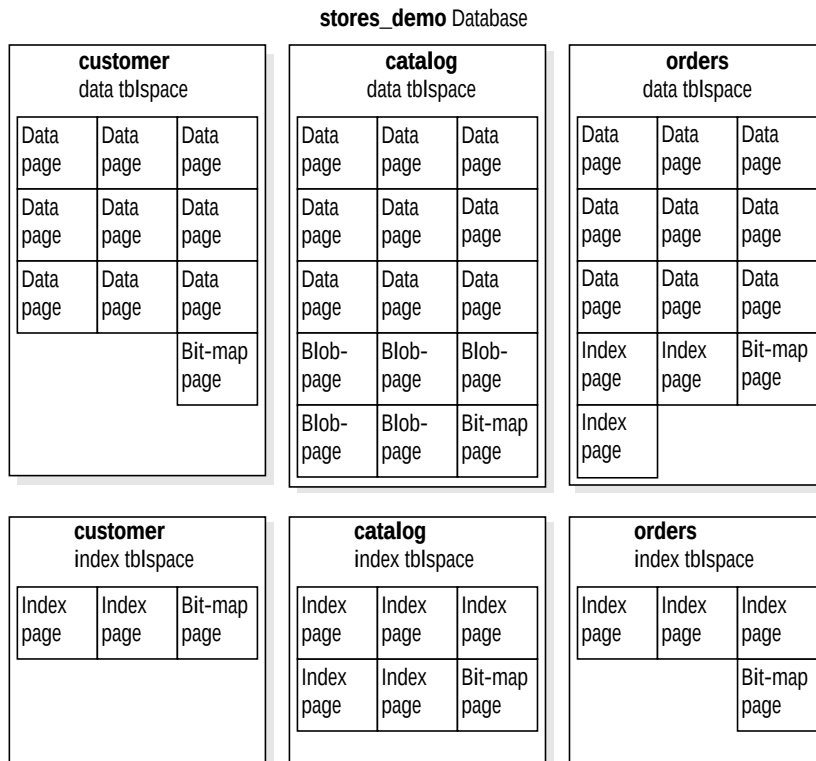
- Pages allocated to data
- Pages allocated to indexes
- Pages used to store TEXT or BYTE data in the dbspace (but not pages used to store TEXT or BYTE data in a blob space)
- Bit-map pages that track page use within the table extents

The index *tblspace* contains the following types of pages:

- Pages allocated to indexes
- Bit-map pages that track page use within the index extents

Figure 14-16 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 14-16
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 14-17](#) 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 14-17
Three Extents That Belong to Two Different Tbspaces in a Single Dbspace

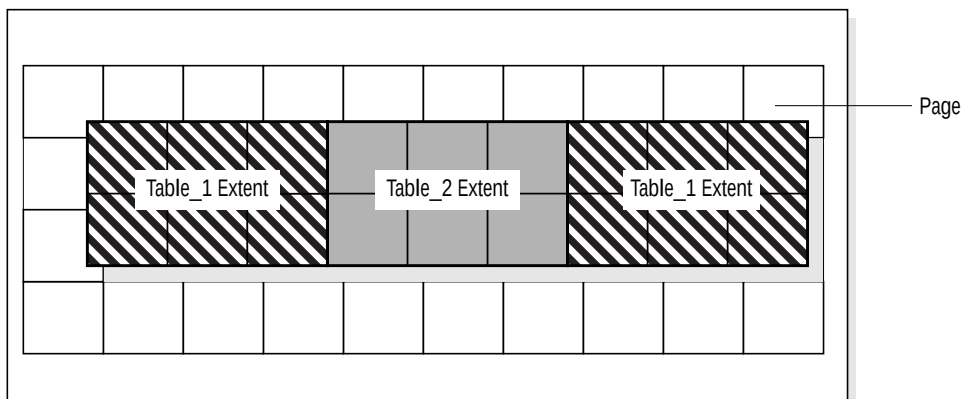


Table Fragmentation and Data Storage

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 [“Fragmentation” on page 16-3](#).

Amount of Disk Space Needed to Store Data

To determine how much disk space you need, follow these steps:

1. Calculate the size requirements of the root dbspace.
2. Estimate the total amount of disk space to allocate to all the database server databases, including space for overhead and growth.

The following 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- and logical-log files
- Temporary tables
- Data
- ON-Archive catalog data
- Control information ♦

You need not store the physical log, the logical log, or the temporary tables in the root dbspace. Include calculations for these items only if you plan to continue to store them in the root dbspace.

If you plan to move the physical and logical logs, the initial configuration for the root dbspace might differ markedly from the final configuration. You can resize the root dbspace after you remove the physical and logical logs. However, the root dbspace must be large enough for the minimum size configuration during disk initialization.

The sections that follow discuss each storage structure in the root dbspace.

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 21-6](#).

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 [“Size and Number of Logical-Log Files” on page 19-7](#).

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that the database server might require for implicit temporary tables. [“Temporary Tables” on page 14-38](#) contains a list of statements that require temporary space. 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 database server creates implicit temporary files when you perform a warm restore. The largest implicit temporary file 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 multiplying the value of LOGSIZE by LOGFILES. For more information on these configuration parameters, see [“Size and Number of Logical-Log Files” on page 19-7](#).

You must also analyze end-user applications to estimate the amount of disk space that the database server might require for explicit temporary tables. See [“Temporary Tables” on page 14-38](#).

By default, the database server stores both implicit and explicit temporary tables in the root dbspace. However, if you decide not to store your temporary tables in the root dbspace, you can use the **DBSPACETEMP** environment variable and configuration parameter to specify a list of dbspaces for temporary files and tables. See [“Storage of Temporary Tables” on page 14-40](#).

Critical Data

Next, decide if users store databases or tables in the root dbspace. If the root dbspace is the only dbspace that you intend to mirror, place all critical data there for protection. Otherwise, store databases and tables in another dbspace.

Estimate the amount of disk space, if any, that you need to allocate for tables stored in the root dbspace.

UNIX

ON-Archive Catalog Data

If you use ON-Archive to perform dbspace and logical-log backups, include space estimates for ON-Archive catalog data. For a description of the tables that compose the ON-Archive catalog, refer to your [Archive and Backup Guide](#). For instructions on how to calculate the size of each table, see your [Performance Guide](#).

UNIX

Control Information

The total amount of disk space required for the database server control information is 3 percent of the size of the root dbspace (sum of physical and logical log, temporary space, and data) plus 25 pages, expressed as kilobytes (or 25 times the database server page size).

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):

1. Decide how many databases and tables you need to store. Calculate the amount of space required for each one.
2. Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
3. 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 between 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.

The sections that follow discuss various strategies for meeting disk-layout goals.

Dbspace 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.

Strive to Associate Partitions with Chunks

When you allocate disk space (buffered or unbuffered files), you allocate it in chunks. A dbspace or a blobspace is associated with one or more chunks. You must allocate at least one chunk for the root dbspace.

Informix recommends that you format your disks so that each chunk is associated with its own disk partition. You can easily track disk-space use when you define every chunk as a separate partition (or device). You can also avoid errors caused by miscalculated offsets.

A disk that is already partitioned might require the use of offsets. For details, see [“Allocating Raw Disk Space on UNIX” on page 15-10](#).

Mirror Critical Data Dbspaces

Mirror the critical dbspaces: the root dbspace, the dbspace that contains the physical log, and the dbspace that contains the logical-log files. You specify mirroring on a chunk-by-chunk basis. Locate the primary and the mirrored chunk on different disks. Ideally, different controllers handle the different disks. [Figure 14-18](#) shows a primary chunk and its mirror.

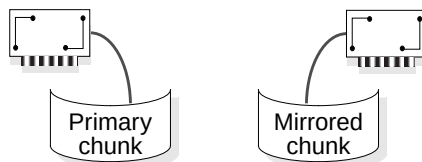


Figure 14-18
*Ideal Disk Layout for
Primary Chunk and
Associated Mirrored
Chunk*

Spread Temporary Storage Space Across Multiple Disks

You can use the **DBSPACETEMP** environment variable and configuration parameter to store a list of dbspaces used for temporary storage. The list can include both temporary and standard dbspaces. To achieve load balancing, design the list so that your temporary disk space is spread across multiple disks. For instructions on how to set DBSPACETEMP, see the chapter on configuration parameters in the *Administrator's Reference*.

Move the Logical and Physical Logs from the Root Dbspace

Whether or not databases use transaction logging, the logical log and physical log both contain data that the database server accesses frequently. Reserved pages are also accessed frequently; they contain internal tables that describe and track all dbspaces, blobspaces, chunks, databases, and tblspaces.

When you allocate disk space (buffered or unbuffered files), you allocate it in chunks. A dbspace or a blobspace is associated with one or more chunks. You must allocate at least one chunk for the root dbspace.

By default, the database server stores the logical and physical logs together with the reserved pages in the root dbspace. Storing the logical and physical logs together is convenient if you have a small, low-volume transaction-processing system. However, maintaining these files together in the root dbspace can become a source of contention as your database system grows.

To reduce this contention and provide better load balancing, move the logical and physical logs to separate partitions or, even better, separate disk drives. For optimum performance, consider creating two additional dbspaces: one for the physical log and one for the logical log. When you move the logs, avoid storing them in a dbspace that contains high-use tables. Instead, consider storing them in a dbspace dedicated to storing only the physical or logical log. For more advice on where to store your logs, see [“Location of the Physical Log” on page 21-9](#) and [“Location of Logical-Log Files” on page 19-10](#).

For instructions on how to change the location of the logical and physical log, see [“Using onparams to Change Physical-Log Location or Size” on page 22-5](#) and [“Moving a Logical-Log File to Another Dbspace” on page 20-7](#).

Consider Account Backup-and-Restore Performance

When you plan your disk layout, consider how the configuration that you choose affects your backup-and-restore procedure. This section describes two configurations that can have a significant impact on your backup-and-restore procedure.

Cluster Catalogs with the Data That They Track

When a disk that contains the system catalog of a particular database fails, the entire database remains inaccessible until you restore the system catalog. Informix recommends that you do not cluster the system catalog tables for all databases in a single dbspace but instead place the catalogs with the data that they track.

Reconsider Separating the Physical and Logical Logs

Although it makes sense from a performance perspective to separate the root dbspace from the physical and logical logs, and the two logs from one another, this configuration is the least desirable in terms of recovery.

Whenever a disk that contains critical information (the root dbspace, physical log, and logical log) fails, the database server comes off-line. In addition, the database server administrator must restore all the database server data, starting in off-line mode, from a level-0 backup before processing can continue.

When you separate the root dbspace from the physical- and logical-log files, you increase the probability that, if a disk fails, it is one that contains critical information (either the root dbspace, physical log, or logical log). For information on how to fragment to improve backup and restore characteristics, see your [*Performance 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. For a discussion of how to optimize your disk layout using table fragmentation, refer to your [Performance Guide](#).

Isolate High-Use Tables

You can place a table with high I/O activity on a disk device dedicated to its use and thus reduce contention for the data stored in the table. When disk drives have different performance levels, you can put the tables with the highest frequency of use on the fastest drives. Placing two high-use tables on separate disk devices reduces competition for disk access when joins are formed between the two tables or when the two tables experience frequent, simultaneous access from multiple applications.

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`. [Figure 14-19](#) illustrates this strategy by showing optimal placement of three frequently used tables.

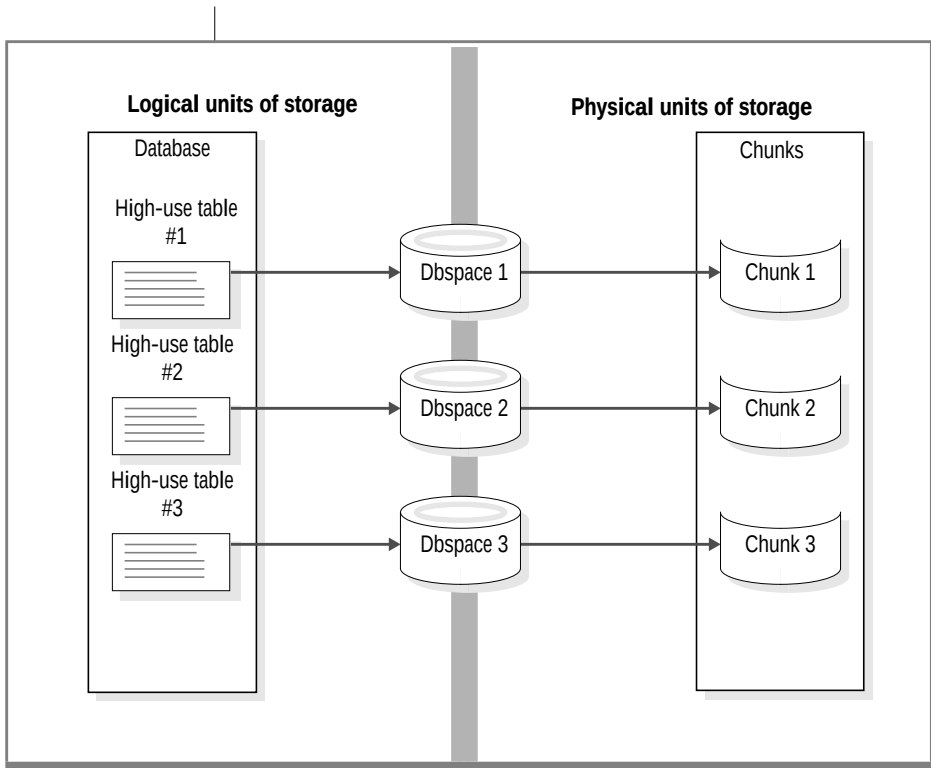


Figure 14-19
Example of High-Use Table Isolation

To take this strategy a step further, fragment a high-use table over multiple disk devices. If you choose an appropriate distribution scheme, the database server routes queries to the appropriate fragment, thereby reducing contention on any single fragment. For more information, see your [Performance Guide](#).

If you have doubts whether spreading your tables across multiple disks can improve performance for your particular configuration, run the **-g iof** option of **onstat**. This option displays the level of I/O operations against each chunk. For details about **onstat**, see the utilities chapter in the *Administrator's Reference*.

Consider Mirroring

You can mirror critical tables and databases to maximize availability. You specify mirroring on a chunk-by-chunk basis. Locate the primary and mirrored chunks for critical tables on different disks. Ideally, different controllers handle the different disks.

Fragmentation gives you a higher level of control over this process. That is, you can mirror chunks that contain specific table fragments. For more information, see your [Performance Guide](#).

Group Tables with Backup and Restore in Mind

When you decide where to place your tables, keep in mind that if a device containing a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. The accessibility (or inaccessibility) of dbspace might influence which tables you group together in a particular dbspace.

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. This situation might influence which dbspace you use to store critical information. If you use ON-Bar for your backup and restore tool, refer to your *Backup and Restore Guide* for more information. If you use ON-Archive or **ontape** for your backup and restore tool, refer to your [Archive and Backup Guide](#) for more information.

Fragmentation gives you greater granularity of backup and restore. When you fragment a table, you can still access the fragments located in the other dbspaces in the event of a dbspace failure. For more information, see your [Performance Guide](#).

Place High-Use Tables on Middle Partition of Disk

To minimize disk-head movement, place the most-frequently accessed data in partitions as close to the middle of the disk as possible. See [Figure 14-20](#). When a disk device is partitioned, the central partitions generally experience the fastest access time. Place the least-frequently used data on the outermost or innermost partitions. This overall strategy minimizes disk-head movement.

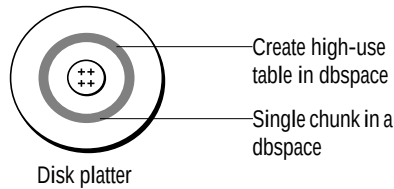


Figure 14-20
Disk Platter with
High-Use Table
Located on Middle
Partitions

To place high-use tables on the middle partition of the disk, create a chunk using raw disk space that is composed of cylinders that reside midway between the spindle and the outer edge of the disk. Then create a dbspace with this same chunk as the initial and only chunk. When you create your high-use tables, use the IN clause of the CREATE TABLE statement to place them in the newly created dbspace.

For information about using raw disk space, see [“Unbuffered or Buffered Disk Access on UNIX” on page 14-7](#) and [“Unbuffered or Buffered Disk Access on Windows NT” on page 14-6](#).

Optimize Table-Extent Sizes

When two or more large, growing tables share a dbspace, their new extents can become interleaved. (See [“Tbspaces” on page 14-42](#).) This interleaving creates gaps between the extents of any one table. (See [Figure 14-17 on page 14-44](#).) Performance might suffer if disk seeks must span more than one extent. Work with the table owners to optimize the table extent sizes and thus limit head movement. For advice on how to alleviate this problem, see your [Performance Guide](#). You can also consider placing the tables in separate dbspaces.

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	1.5 gigabytes	No
Disk 2	2 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.

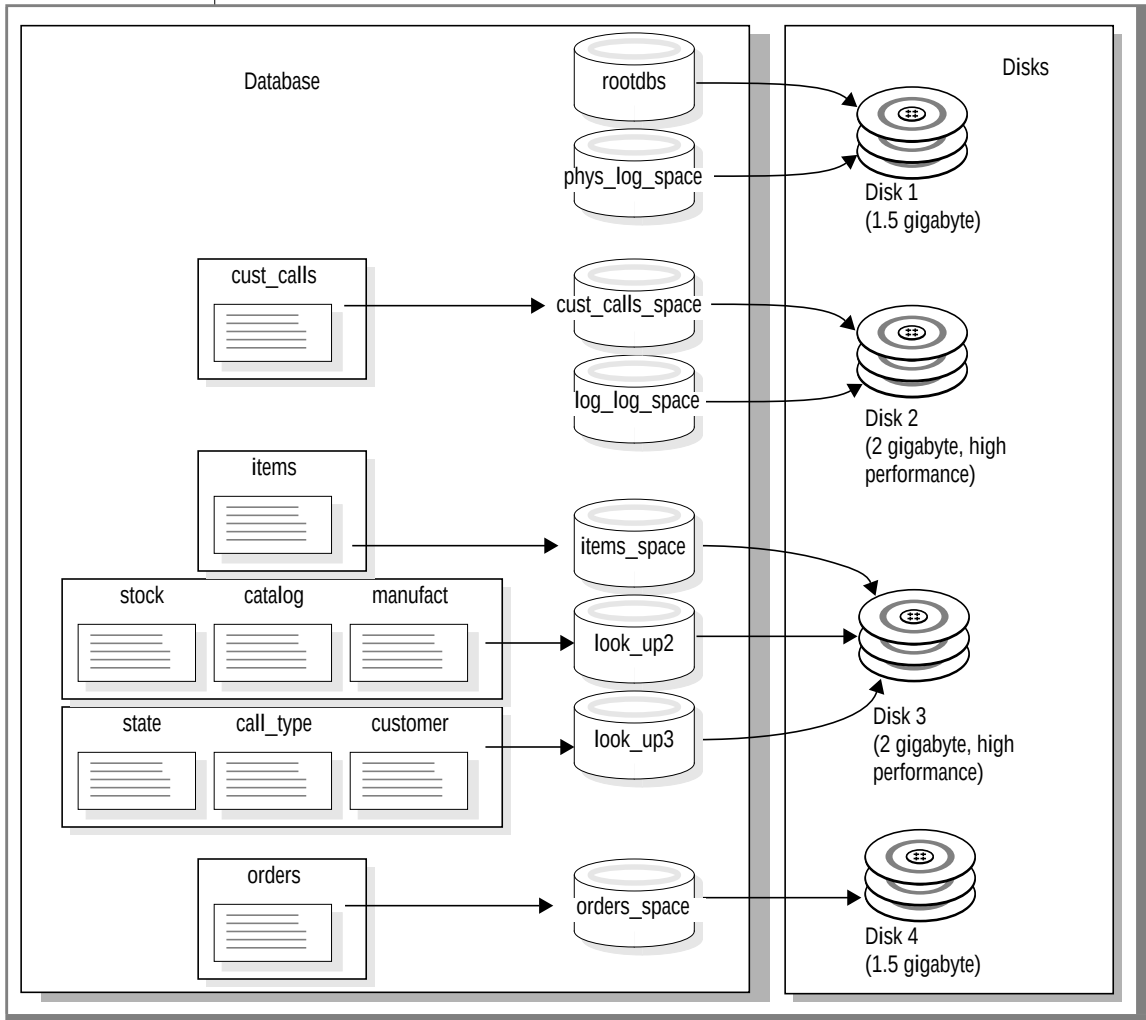
Table Name	Maximum Size	Access Rate
cust_calls	1.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample Layout When Performance Is Highest Priority

Figure 14-21 on page 14-58 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 14-21
Disk Layout Optimized for Performance

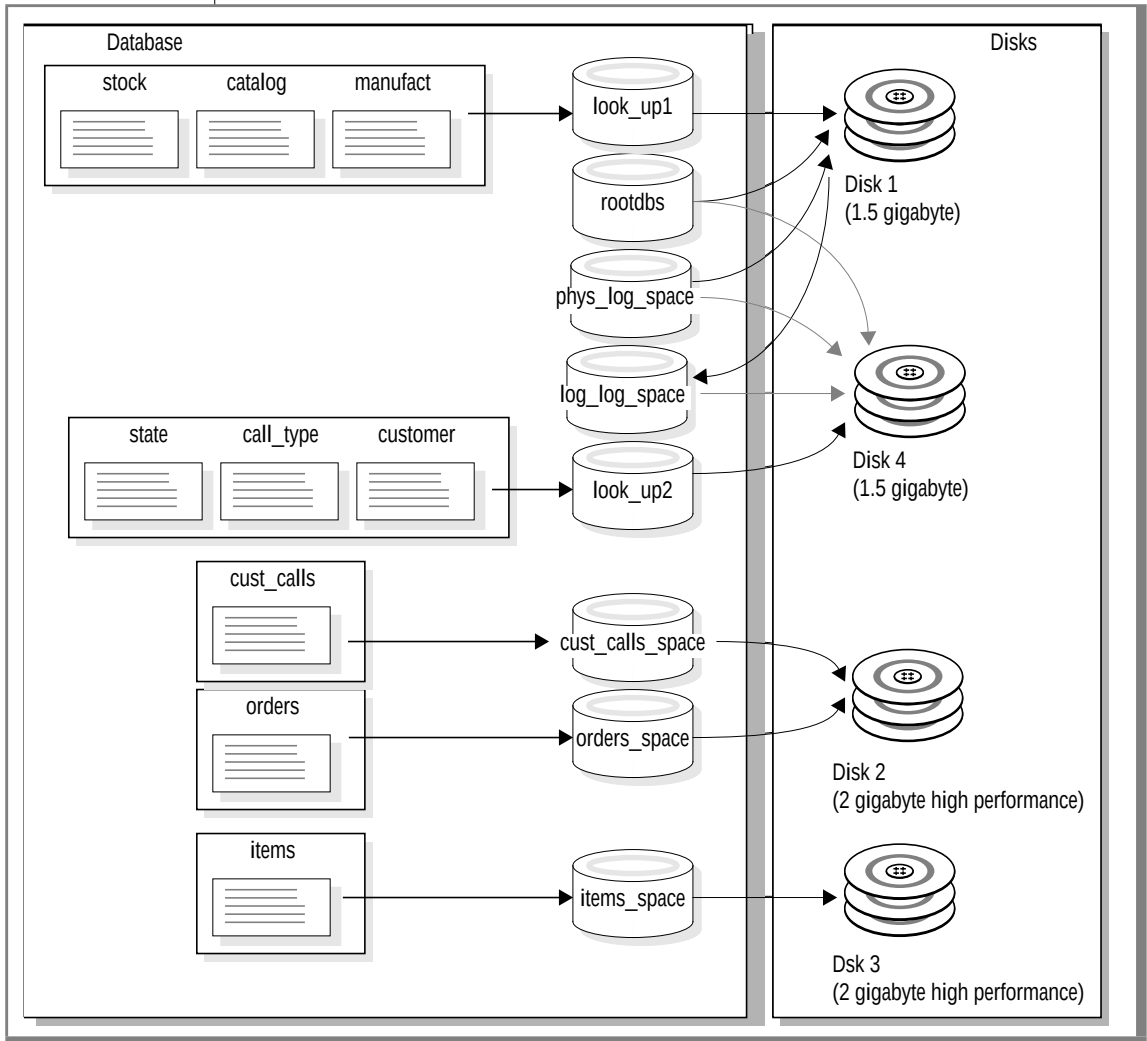


Sample Layout When Availability Is Highest Priority

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 14-22 on page 14-60](#). 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 14-22
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 2 gigabytes, 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 2 gigabytes.

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. For more information, see [“Place High-Use Tables on Middle Partition of Disk” on page 14-55](#). (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 14-20](#).)

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. This strategy helps reduce contention between programs that access the same table, as explained in [“Place High-Use Tables on Middle Partition of Disk” on page 14-55](#).

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 24-6](#).

Figure 14-23 illustrates the role of fragments in specifying the location of data.

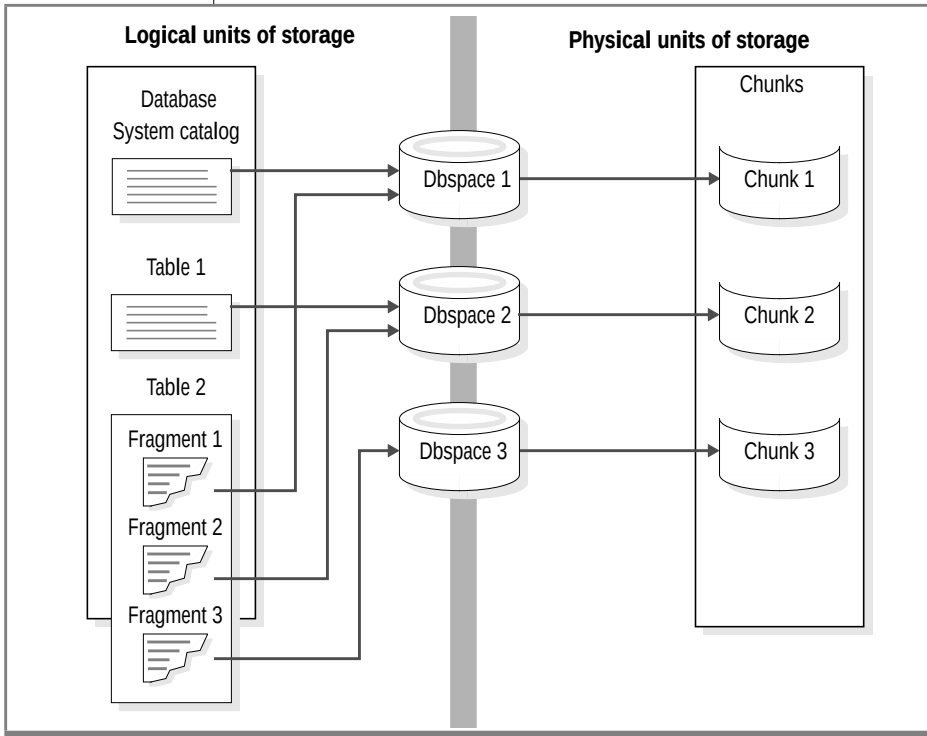


Figure 14-23
*Dbspaces That Link
Logical Units
(Including Table
Fragments) and
Physical Units of
Storage*

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  
dbospace1, dbospace2, dbospace3;
```

```
CREATE TABLE tablename ...FRAGMENT BY EXPRESSION  
    <Expression 1> in dbospace1,  
    <Expression 2> in dbospace2,  
    <Expression 3> in dbospace3;
```

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 [Informix Guide to Database Design and Implementation](#)

Managing Disk Space

In This Chapter	15-5
Initializing Disk Space.	15-6
Allocating Disk Space	15-6
Specifying an Offset	15-7
Specifying an Offset for the Initial Chunk of Root Dbspace	15-8
Specifying an Offset for Additional Chunks	15-8
Using Offsets to Create Multiple Chunks	15-8
Allocating a File for Disk Space on UNIX.	15-9
Allocating a File for Disk Space on Windows NT	15-10
Allocating Raw Disk Space on UNIX	15-10
Allocating Raw Disk Space on Windows NT.	15-12
Creating Standard Device Names	15-12
Backing Up After You Change the Physical Schema	15-13
Creating a Dbspace.	15-13
Specifying Pathnames for Dbspaces.	15-14
Specifying Names and Maximum Number of Storage Spaces	15-15
Backing Up the New Dbspace.	15-15
Creating a Temporary Dbspace	15-15
Creating a Dbspace with onspaces	15-16
Creating a Dbspace with ON-Monitor	15-17
Adding a Chunk to a Dbspace	15-17
Backing Up the New Chunk	15-18
Naming Chunks and Storage Spaces	15-18
Limiting Chunk Size and Number	15-19
Adding a Chunk with onspaces.	15-19
Adding a Chunk with ON-Monitor	15-19

Creating a BlobSpace	15-20
Backing Up the New BlobSpace	15-21
Determining Blobpage Size.	15-21
Determining Database Server Page Size	15-22
Creating a BlobSpace with onspaces.	15-23
Creating a BlobSpace with ON-Monitor	15-23
Adding a Chunk to a BlobSpace	15-24
Creating an Sbspace	15-24
Backing Up the New Sbspace	15-25
Sizing Sbspace Metadata.	15-25
Sizing the Metadata Area Automatically	15-26
Estimating the Size of the Metadata Area	15-26
Adding a Chunk to an Sbspace.	15-27
Loading Data Into a Table	15-28
Altering Storage Characteristics of Smart Large Objects	15-28
Dropping a Chunk	15-29
Verifying Whether a Chunk Is Empty	15-29
Dropping a Chunk from a DbSpace with onspaces.	15-30
Dropping a Chunk from a BlobSpace	15-30
Dropping a Chunk from an Sbspace with onspaces	15-30
Dropping a Storage Space	15-31
Preparing to Drop a Storage Space	15-31
Backing Up After Dropping a Storage Space	15-32
Dropping a Mirrored Storage Space	15-32
Dropping a Storage Space with onspaces	15-32
Dropping a DbSpace or BlobSpace with ON-Monitor	15-33
Optimizing BlobSpace Blobpage Size.	15-33
Determining BlobSpace Storage Efficiency	15-34
Obtaining BlobSpace Storage Statistics	15-34
Determining Blobpage Fullness with oncheck -pB	15-35
Interpreting Blobpage Average Fullness	15-36
Applying Efficiency Criteria to Output	15-37

Managing Extspaces	.15-37
Creating an Extspace	.15-37
Dropping a Extspace	.15-38
Skipping Inaccessible Fragments.	.15-38
Using the DATASKIP Configuration Parameter	.15-39
Using the Dataskip Feature of onspaces	.15-39
Using onstat to Check Dataskip Status	.15-39
Using the SQL Statement SET DATASKIP	.15-40
Effect of the Dataskip Feature on Transactions.	.15-41
Determining When to Use Dataskip	.15-41
Determining When to Skip Selected Fragments	.15-42
Determining When to Skip All Fragments	.15-42
Monitoring Fragmentation Use	.15-42
Displaying Databases	.15-43
Using SMI Tables.	.15-43
Using ON-Monitor	.15-44
Monitoring the Database Server for Disabling I/O Errors	.15-44
Using the Message Log to Monitor Disabling I/O Errors	.15-44
Using Event Alarms to Monitor Disabling I/O Errors	.15-45
Monitoring Disk Usage	.15-46
Monitoring Chunks	.15-46
Using Command-Line Utilities.	.15-46
Using ON-Monitor	.15-51
Using SMI Tables	.15-52
Monitoring Tblspaces and Extents	.15-53
Using Command-Line Utilities.	.15-53
Using SMI Tables	.15-55
Using System Catalog Tables	.15-56
Monitoring Simple Large Objects in a Blobspace	.15-56
Using Command-Line Utilities.	.15-56
Using ON-Monitor	.15-61
Monitoring Simple Large Objects in a Dbospace	.15-61
Monitoring Sbspaces	.15-62
Features Beyond the Scope of the Database Server.	.15-63
No Bad-Sector Mapping	.15-63
No Compression of TEXT and BYTE Data Types	.15-63

In This Chapter

This chapter provides the instructions that you need to manage effectively the disk spaces and data that the database server controls. It assumes you are familiar with the terms and concepts contained in [Chapter 14, “Data Storage.”](#)

This chapter covers the following topics:

- Initializing disk space
- Allocating disk space
- Setting configuration variables related to disk management
- Backing up after you change the physical schema
- Managing chunks and storage spaces
 - Creating and dropping dbspaces, blobspaces, and sbspaces
 - Allocating, adding, and dropping chunks from dbspaces, blobspaces, and sbspaces
 - Optimizing blobpage size
 - Creating and dropping extspaces
- Skipping inaccessible fragments
- Monitoring disk space
- Monitoring simple-large-object data and smart-large-object data
- Features beyond the scope of the database server

Your *Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents and how to reclaim space in an empty extent.



Initializing Disk Space

Disk-space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk and to initialize shared memory. When you initialize disk space, shared memory is automatically initialized for you as part of the process.

Typically, you initialize disk space just once in the life of a database server. This action occurs when you bring the database server on-line for the first time.

Warning: When you initialize the database server disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all data in the earlier database server instance becomes inaccessible and, in effect, is destroyed.

For information on initializing the database server, see [“Initializing Disk Space” on page 8-4](#).

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 14-7](#)
- [“Amount of Disk Space Needed to Store Data” on page 14-45](#)
- [“Disk-Layout Guidelines” on page 14-48](#)

Before the database server can use disk space, you might need to perform these tasks:

- Initializing disk space
- Creating a dbspace or blobspace
- Adding a chunk to an existing dbspace or blobspace
- Mirroring an existing dbspace or blobspace

You can allocate either an empty file or a portion of raw disk for database server disk space.

UNIX

On UNIX, if you allocate raw disk space, Informix recommends 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 Standard Device Names” on page 15-12](#).

Using a UNIX file and its inherent operating-system interface for database server disk space also is referred to as using *cooked space*. ♦

WIN NT

On Windows NT, Informix usually recommends that you use NTFS files for database server disk space. For more information on this recommendation, see [“Unbuffered or Buffered Disk Access on Windows NT” on page 14-6](#). ♦

Specifying an Offset

When you allocate a chunk of disk space to the database server, you might want to 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

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. (For convenience, it is referred to here as the VTOC.) The VTOC is commonly stored on the first track of the drive. A table of alternate sectors and bad-sector mappings (also called 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 with one of the following utilities:

- **onspaces**
- **ON-Monitor** ♦

For more information on specifying an offset for chunks of database server space, see [“Creating a Dbspace” on page 15-13](#).

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.

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 a File for Disk Space on UNIX

To allocate a file for database server disk space on UNIX, log in as user **informix** and concatenate null to the filename that the database server will use for disk space. The file should have permissions set to 660 (rw-rw----). Group and owner must be set to **informix**. [Figure 15-1](#) illustrates these steps and allocates the file **/usr/data/my_chunk** for disk space.

Figure 15-1
Preparing Cooked File Space

Step	Command	Comments
1.	su informix	Log in as user informix . (Enter the password.)
2.	cd /usr/data	Change directories to the directory where the cooked space will reside.
3.	cat /dev/null > my_chunk	Create your chunk by concatenating null to a file (in this example, a file named my_chunk).
4.	chmod 660 my_chunk	Set the permissions of the file to 660 (rw-rw----).
5.	ls -lg my_chunk -rw-rw---- 1 informix informix 0 Oct 12 13:43 my_chunk	Use <code>ls -l</code> if you are using System V UNIX. Verify that both group and owner of the file are informix . You should see something like this line (which has wrapped around).

For information on how to create a dbspace or blobspace using the file you have allocated, refer to [“Creating a Dbspace” on page 15-13](#) and [“Creating a Blobspace” on page 15-20](#).

WIN NT

Allocating a File for Disk Space on Windows NT

On Windows NT, 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, the first step is to create a null (zero bytes) file. Perform the following steps to allocate NTFS file space for a dbspace or blobspace, including a mirror dbspace or blobspace:

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 15-13](#) and [“Creating a Blobspace” on page 15-20](#).

You must also follow the preceding steps prior to adding a chunk to a dbspace or blobspace.

UNIX

Allocating Raw Disk Space on UNIX

For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation.

In general, to create raw disk space, you can either repartition your disks or unmount an existing file system. In either case, take proper precautions to back up any files before you unmount the device. (See [“Unbuffered or Buffered Disk Access on UNIX” on page 14-7](#).)

Change the group and owner of the character-special devices to **informix**. The filename of the character-special device usually begins with the letter *r*.



Verify that the operating-system permissions on the character-special devices are `crw-rw----`.

Warning: After you create the raw device that the database server uses for disk space, carefully heed the following warnings:

- Do not create file systems on the same raw device that you allocate for the database server disk space.
- Do not use the same raw device as swap space that you allocate for the database server disk space.

Create a link between the character-special device name and another filename with the UNIX link command, usually **ln**.

The link enables you to replace quickly the disk where the chunk is located. The convenience becomes important if you need to restore your database server data. The restore process requires all chunks that were accessible at the time of the last dbspace backup to be accessible when you perform the restore. The link means that you can replace a failed device with another device and link the new device pathname to the same filename that you previously created for the failed device. You do not need to wait for the original device to be repaired.

Execute the UNIX command **ls -lg (ls -l on System V UNIX)** on your device directory to verify that both the devices and the links exist. The following example shows links to raw devices. If your operating system does not support symbolic links, hard links will work as well.

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

WIN NT

Allocating Raw Disk Space on Windows NT

To allocate raw disk space on Windows NT, specify a physical drive or a disk partition for **onspaces**, rather than a filename, when you create a dbspace or add a chunk to a dbspace. The next example adds a chunk of 5000 kilobytes of raw disk space, at an offset of 5200 kilobytes, to dbspace **dpssc3**.

```
onspaces -a dbssc3 \\.\e: -o 5200 -s 5000
```

You must be a member of the **Informix-Admin** group when you create a dbspace or add a chunk to a dbspace. The raw disk space can be formatted or unformatted disk space.



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 are responsible to ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.*

UNIX

Creating Standard Device Names

On UNIX, use symbolic links to assign abbreviated standard device names. 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 work as well.

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

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.
- add, move, drop, or resize 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 sbpace.
- add, move, or drop a chunk to a dbspace, blobspace, or sbpace.

You must perform a level-0 backup of the modified storage spaces to ensure that you can restore the data when you convert a nonlogging database to a logging database.

Creating a Dbspace

This section explains how to create a standard dbspace and a temporary dbspace. See [“Dbspaces” on page 14-19](#) and [“Temporary Dbspaces” on page 14-22](#).

You can use the following utilities to create a dbspace:

- **onspaces**
Use **onspaces** to manage sbspaces and extspaces.
- **ON-Monitor**
ON-Monitor supports operations on dbspaces and blobspaces. ♦

Before you create a dbspace, you must first allocate disk space as described in [“Allocating Disk Space” on page 15-6](#).

On UNIX, you must be logged in as user **informix** or **root** to create a dbspace. On Windows NT, users in the **Informix-Admin** group can create a dbspace.

If you are creating a standard dspace, the database server can be in on-line mode. 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.

When the initial chunk of the dspace that you are creating is a cooked file on UNIX or an NTFS file on Windows NT, 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 dspace 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 dspace. Remove this file to reclaim the space.

Specifying Pathnames for Dbspaces

Specify an explicit pathname for the initial chunk of the dspace as follows:

- If you are using raw disks on UNIX, Informix recommends that you use a linked pathname. (See [“Creating Standard Device Names” on page 15-12.](#)) ♦
- [“Creating Standard Device Names” on page 15-12](#) If you are using raw disks on Windows NT, 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.

UNIX

WIN NT

Specifying Names and Maximum Number of Storage Spaces

Specify a dbspace 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.

You can create a maximum of 2048 storage spaces on the system in any combination of the following storage spaces:

- Dbspaces
- Blobspaces
- Sbspaces

Backing Up the New Dbspace

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

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 **-t** option.
For more information, refer to [“Creating a Dbspace with onspaces” on page 15-16](#).
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.

Creating a Dbspace with onspaces

To create a dbspace with **onspaces**, use the **-c** option of **onspaces** as shown in the following example.

This example creates a 10-megabyte mirrored dbspace, **dbspce1**, using raw disk space on UNIX. An offset of 5000 kilobytes is specified for both the primary and mirrored chunks.

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10000 -m /dev/raw_dev2 5000
```

The following example creates a 5000 kilobyte dbspace, **dbspc3**, with an offset of 200 kilobytes, from raw disk space (drive e:) on Windows NT:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5000
```

The following example creates a 5-megabyte temporary dbspace named **temp_space**:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5000
```

For reference information on creating a dbspace with **onspaces**, see the utilities chapter in the *Administrator's Reference*.

UNIX

Creating a Dbspace with ON-Monitor

To create a dbspace with ON-Monitor, follow these instructions:

1. Select the Create option from the Dbspaces menu.
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*.

Adding a Chunk to a Dbspace

If one of your dbspaces is becoming full, you might want to add a new chunk. Before you do, however, you must first allocate disk space as described in [“Allocating Disk Space” on page 15-6](#).

You add a *chunk* when you need to increase the amount of disk space allocated to a storage space. To add a chunk on UNIX, you must be logged in as user **informix** or **root**. To add a chunk on Windows NT, you must be a member of the **Informix-Admin** group.

If you are adding a chunk to a mirrored storage space, you must also add a mirrored chunk.

UNIX



When you add a chunk that is allocated as a cooked file on UNIX or an NTFS file on Windows NT, the database server verifies that the disk space is sufficient for the new chunk by creating and then removing a file of the size requested. If the size of the chunk is greater than the available space on the disk, the database server might inadvertently fill your file system in the process of verifying available disk space.

To add a chunk to a dbspace, use one of the following utilities.

- onspaces
- ON-Monitor ♦

Important: *You can add a chunk while the database server is in on-line or quiescent mode. The newly added chunk (and its associated mirror, if one exists) is available immediately.*

Backing Up the New Chunk

After you create the new chunk, you must perform a level-0 backup of the root dbspace and the dbspace, blobspace, or sbpace that contains the chunk.

Naming Chunks and Storage Spaces

You must specify an explicit pathname for the chunk. For more information, see [“Creating a Dbspace” on page 15-13](#).

Use these naming rules when you create storage spaces or add a chunk. The name 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

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.

Limiting Chunk Size and Number

On most platforms, the maximum size of a chunk is 2 gigabytes, but on some platforms the maximum chunk size is 4 gigabytes. To determine which chunk size your platform supports, refer to your machine notes file.

You can allocate a maximum of 2048 chunks and 2048 storage spaces on a given database server system. The storage spaces can be any combination of storage spaces.

Adding a Chunk with onspaces

To add a chunk to a dbspace, use the **-a** option of **onspaces** as illustrated in the following example. This 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 10000 -m /dev/raw_dev2 200
```

The next example adds a chunk of 5000 kilobytes of raw disk space, at an offset of 5200 kilobytes, to dbspace **dbspc3**.

```
onspaces -a dbspc3 \\.e: -o 5200 -s 5000
```

For reference information on adding a chunk to a dbspace with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

Adding a Chunk with ON-Monitor

To add a chunk to a dbspace, follow these instructions:

1. Choose the Dbspaces menu, Add_chunk option.
2. Use RETURN or the arrow keys to select the blobspace or dbspace that will receive the new chunk and press CTRL-B or F3.
3. The next screen indicates whether the blobspace or dbspace is mirrored. If it is, enter Y in the **Mirror** field.
4. If the dbspace to which you are adding the chunk is a temporary dbspace, enter Y in the **Temp** field.

5. If you indicated that the dbspace or blobSpace is mirrored, you must specify both a primary chunk and 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 mirrored-chunk complete pathname, size, and optional offset in the mirror-chunk section of the screen.

Creating a BlobSpace

Before you create a blobSpace, you must first allocate disk space as described in [“Allocating Disk Space” on page 15-6](#). Before you create a blobSpace, take time to determine what blobSpace size is optimal for your environment. For instructions, see [“Optimizing BlobSpace Blobpage Size” on page 15-33](#).

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.

Specify an explicit pathname for the initial chunk of the blobSpace. For more information, see [“Creating a Dbspace” on page 15-13](#).



Important: *You can create a blobSpace while the database server is in on-line or quiescent mode.*

You can mirror the blobSpace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.

To create a blobSpace on UNIX platforms, you must be logged in as user **informix** or **root**. To create a blobSpace on Windows NT platforms, you must be a member of the **Informix-Admin** group. You can use one of the following tools to create a blobSpace:

- **onspaces**
- ON-Monitor ♦

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 19-26.](#))

When the initial chunk of the blobspace that you are creating is a file, the database server verifies that 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 blobspace is created. However, the file that the database server creates for the initial chunk is not removed. Its size represents the space left on your file system before you attempted to create the blobspace. You must remove this file to reclaim the space.

Backing Up the New Blobspace

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

Determining Blobpage Size

When you create a blobspace, try to create a blobpage size that approximates the size of the most frequently occurring TEXT or BYTE data that the blobspace holds. For example, if you are storing 160 simple large objects, and you expect 120 simple large objects to be 12 kilobytes and 40 simple large objects to be 16 kilobytes, a 12-kilobyte blobpage size stores the objects most efficiently. This configuration allows the majority (120) of the objects to be stored in a single blobpage, while the other 40 objects require two blobpages each. Even though 8 kilobytes are wasted in the second blobpage (8 kilobytes wasted space per page * 40 pages = 320 kilobytes total wasted space), this arrangement does not waste as much space as using a blobpage size of 16 kilobytes (4 kilobytes wasted space per page * 120 pages = 480 kilobytes total wasted space).

In some circumstances, you might want to use the larger, 16-kilobyte blobpage size. If speed and reducing the number of locks are primary concerns, use a 16-kilobyte blobpage so that every simple large object can be stored on a single blobpage. You also might want to use a 16-kilobyte blobpage size if those blobpages are accessed more frequently than the 12-kilobyte blobpages.

To continue the example, assume that your database server page size is 2 kilobytes. If you decide on a 12-kilobyte blobpage size, specify the blobpage size parameter as 6 (pages). If your database server page size is 4 kilobytes, specify the blobpage size parameter as 3 (pages). In general, divide the size of the simple large object (rounded up to the nearest kilobyte) by the page size to determine the blobpage size parameter.

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 blobpage, each with an appropriately sized blobpage. See [“Tables” on page 14-37](#).

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.
- Select the Shared-Memory option of the ON-Monitor Parameters menu. ON-Monitor displays a list of shared-memory parameters. The database server page size is the last entry on the page.
- Select the Initialize option of the ON-Monitor Parameters menu. The first entry displayed on the screen is the page size. ♦

Creating a Blobspace with onspaces

To create a blobspace with **onspaces**, use the **-c** option as illustrated in the following example. This 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 10000 -m /dev/raw_dev2 200
```

For reference information on creating a blobspace with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

UNIX

Creating a Blobspace with ON-Monitor

To create a blobspace with ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, 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 15-22](#).) 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 mirror-chunk full pathname, size, and optional offset in the mirror-chunk section of the screen.

Adding a Chunk to a BlobSpace

Adding a chunk to a blobSpace is identical to adding a chunk to a dBSpace. Both procedures are explained in [“Adding a Chunk to a DSpace” on page 15-17](#).

Creating an Sbspace

To create an sbspace on UNIX, you must be logged in as user **informix** or **root**. To create an sbspace on Windows NT, you must be a member of the **Informix-Admin** group. You can use the **onspaces** utility to create an sbspace.

Important: *You can create an sbspace while the database server is in on-line or quiescent mode.*

Specify an sbspace 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.

The procedure and considerations for creating an sbspace are similar to those for creating a blobSpace. However, you can specify additional optional parameters with the **onspaces -Df**, **-Mo**, and **-Ms** options:

- The **-Df** option lets you specify various default storage characteristics and other attributes for smart large objects that will be stored in the sbspace.

For more information, refer to [“Storage Characteristics of Sbspaces” on page 14-29](#) and [“Other Attributes of Sbspaces” on page 14-30](#).

- You can specify a metadata area for an sbspace or let the database server calculate the size of the metadata area.

If you want to control the size of the metadata area, use the **onspaces -Mo** and **-Ms** options to specify the size and offset of the sbspace metadata.

If you do not use the **-Mo** and **-Ms** options, the database server estimates the size of the metadata by dividing the estimated average size (the value in the `AVG_LO_SIZE` tag) of the smart large objects by the size of the user data area.



To create an sbspace with **onspaces**, use the **-c** and **-S** options as illustrated in the following example:

```
% onspaces -c -S sbsp4 -p /dev/raw_dev1 -o 500 -s 20000 -m /dev/raw_dev2 500 -Ms  
150 -Mo 200 -Df "AVG_LO_SIZE=32"
```

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. (If you do not specify the **AVG_LO_SPACE -Df** tag, the default average smart-large-object size is 8 kilobytes.)

For information about sizing sbspace metadata, see [“Sizing Sbspace Metadata” on page 15-25](#). For information about creating an sbspace with **onspaces** and default options for smart large objects, see **onspaces** in the utilities chapter of the *Administrator’s Reference*.

Backing Up the New Sbspace

After you create the sbspace, you must perform a level-0 backup of the root dbspace and the new sbspace.

Sizing Sbspace Metadata

The metadata portion of an sbspace grows as smart large objects are added to that sbspace; therefore, it is important to size correctly the metadata area for an sbspace. The first chunk in an sbspace must have a metadata area whether calculated by the database server or the user.

Sizing the Metadata Area Automatically

The easiest way to size the metadata area of an sbspace is to create only one chunk, specifying the average large object size but not the metadata size.

To size the metadata area automatically

1. Estimate the expected average size and number of the smart large objects.
2. Create an sbspace with one chunk.

Use the **onspaces -Df** option `AVG_LO_SIZE` to specify the average size of the smart large objects. If you do not specify `AVG_LO_SIZE`, the database server uses the default average large object size of 64 kilobytes.

You do not need to use the **onspaces -Ms** option because the database server automatically estimates the metadata area for the first chunk and all subsequent chunks by dividing the estimated average size of the smart large objects by the size of the user data area.



Important: *If the sbspace runs out of metadata space, you need to add another chunk to the sbspace. You can either let the database server determine the amount of metadata space or calculate it to ensure you do not run out of metadata space.*

Estimating the Size of the Metadata Area

The formula that the system uses to estimate the size of the metadata area presumes that the metadata area being allocated contains only the smart-large-object header partition for the current chunk. The smart-large-object header partition contains one fourth page (system page size) for each smart large object in a chunk.

If you use the **onspaces -a -U** option to create sbspace chunks that contain user data only, the metadata area in the first chunk might contain the smart-large-object header partition for multiple sbspace chunks. In this case, you can state the size of the metadata area explicitly to ensure that the sbspace does not run out of metadata space.

For information on calculating the size of the metadata area, see estimating pages occupied by smart large objects in the chapter on table performance considerations in your [Performance Guide](#).

Adding a Chunk to an Sbspace

Adding a chunk to an sbspace is identical to adding a chunk to a dbspace. 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:

- If you want 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.
- If you do not use the **-Mo** and **-Ms** options, the database server estimates the size of the metadata by dividing the estimated average size of the smart large objects by the size of the user data area. The database server allocates metadata space to the new chunk.
- If you want to use the chunk for user data only, specify the **-U** option.

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/raw_dev1 -o 200 -s 10000 -m /dev/raw_dev2 200 -U
```

After you add a chunk to the sbspace, you must perform a level-0 backup of the root dbspace and the sbspace.

For reference information on adding a chunk to an sbspace with **onspaces**, see the utilities chapter of the *Administrator's Reference*. For more information on sizing sbspace metadata, see [“Sizing Sbspace Metadata” on page 15-25](#).

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	Reference
DB-Access LOAD statement	Yes	LOAD statement in the Informix Guide to SQL: Syntax
dbload utility	Yes	Informix Migration Guide
dbimport utility	No	Informix Migration Guide
onload utility	No	Informix Migration Guide
Informix ESQL/C programs	Yes	Informix ESQL/C Programmer's Manual
High-Performance Loader (HPL)	Yes	Guide to the High-Performance Loader

Altering Storage Characteristics of Smart Large Objects

When you create an sbspace with the **onspaces -c -S** command, you can use the defaults for the storage characteristics and attributes (such as logging and buffering). These defaults allow you to define an sbspace quickly and start to use it. For more information, refer to “[Storage Characteristics of Sbspaces](#)” on page 14-29 and “[Other Attributes of Sbspaces](#)” on page 14-30.

After you monitor sbspaces, you might want to change the storage characteristics, logging status, lock mode, or other attributes for new smart large objects. For more information on monitoring and altering sbspaces, refer to your [Performance Guide](#).

Dropping a Chunk

Use **onspaces** 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 On-line Mode	Database Server in Quiescent Mode	Database Server in Off-line Mode
Dbspace chunk	Yes	Yes	No
Temporary dbspace chunk	Yes	Yes	No
Blobspace chunk	No	Yes	No
Sbspace chunk	No	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 **dbsp3** on UNIX. An offset of 300 kilobytes is specified.

```
% onspaces -d dbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of a dbspace with the syntax in the previous example. Instead, you must drop the dbspace. Use the **fchunk** column of **onstat -d** to determine which is the initial chunk of a dbspace. For more information about **onstat**, see 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 Dbspace with onspaces” on page 15-30](#) 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 quiescent mode when you drop a chunk from an 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.

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
```

For information on using **onstat** and dropping a chunk from a dbspace with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

Dropping a Storage Space

Use **onspaces** to drop a storage space.

On UNIX, you must be logged in as **root** or **informix** to drop a storage space. On Windows NT, you must be a member of the **Informix-Admin** group to drop a storage space.

Before you drop a storage space, ensure that the database server is in either on-line or quiescent mode. You cannot drop a storage space when it is in off-line 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.

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. You cannot drop the root dbspace.

Before you drop an sbspace, you must drop all tables that have a CLOB or BLOB column that references 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.

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.

If you are using ON-Bar for your backup and restore system, see the *Backup and Restore Guide*. If you are using ON-Archive or the **ontape** utility as your backup and restore tool, refer to your [Archive and Backup Guide](#).



Warning: After you drop a *dbspace*, *blobospace*, or *sbspace*, the newly freed chunks are available for reassignment to other *dbspaces*, *blobospaces*, or *sbspaces*. However, before you reassign the newly freed chunks, you must perform a level-0 backup of the root *dbspace* and the modified storage space. If you do not perform this backup, and you subsequently need to perform a restore, the restore might fail because the *dbspace*-backup reserved pages are not up-to-date.

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 25-13.) This action drops the *dbspace*, *blobospace*, 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 *dbspace* called **blobbsp3** and its mirrors.

```
% onspaces -d blobbsp3
```



UNIX

Use the **-d** option with the **-f** option if you want to drop an sbspace that contains data. If you omit the **-f** option, you cannot drop an sbspace that contains data. This example drops an sbspace called **sbspc4** and its mirrors.

```
% onspaces -d sbspc4 -f
```

Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the deleted smart large objects.

For information on dropping a storage space with **onspaces**, see the utilities chapter of the *Administrator's Reference*.

Dropping a Dbspace or Blobspace with ON-Monitor

To drop a dbspace or blobspace with ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, Drop option.
2. Use RETURN or arrow keys to scroll to the dbspace or blobspace that you want to drop.
3. Press CTRL-B or F3.

You are asked to confirm that you want to drop the dbspace or blobspace.

Optimizing Blobspace Blobpage Size

Familiarize yourself with the database server approach to blobspace storage of TEXT and BYTE data before you begin this section. For background information, see blobspace structure and blobspace page types in the disk structures and storage chapter of the *Administrator's Reference*.

This section is not applicable if you store TEXT and BYTE data in dbspaces.

Determining Blobspace Storage Efficiency

When you are evaluating blobspace storage strategy, you can measure efficiency by two criteria:

- Blobpage fullness
- Blobpages required per simple large object

Blobpage fullness refers to the amount of data within each blobpage. TEXT and BYTE data stored in a blobspace cannot share blobpages. Therefore, if a single simple large object requires only 20 percent of a blobpage, the remaining 80 percent of the page is unavailable for use. However, you want to avoid making the blobpages too small. When several blobpages are needed to store each simple large object, you increase the overhead cost of storage. For example, more locks are required for updates because a lock must be acquired for each blobpage.

Obtaining Blobspace Storage Statistics

To help you determine the optimal blobpage size for each blobspace, use the following database server utility commands:

- **oncheck -pB**
- **oncheck -pe**

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)

The **oncheck -pe** command can provide 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 number of the database server pages used by each table to store its associated TEXT and BYTE data

Determining Blobpage Fullness with oncheck -pB

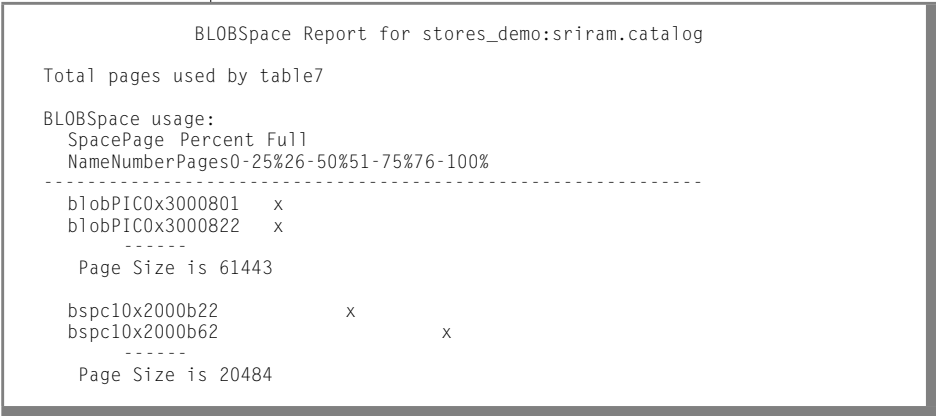
The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. These statistics provide a measure of storage efficiency for individual simple large objects in a database or table. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

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
```

Figure 15-2 shows the output of this command.

Figure 15-2
Output of
oncheck -pB



Space Name is the name of the blobspace that contains one or more simple large objects stored as part of the table (or database).

Page Number is the starting address in the blobspace of a specific simple large object.

Pages is the number of the database server pages required to store this simple large object.

Percent Full is a measure of the average fullness of all the blobpages that hold this simple large objects.

Page Size is the size in bytes of the blobpage for this blobspace. Blobpage size is always a multiple of the database server page size. For instructions on how to obtain the page size for your database server, see [“Determining Database Server Page Size” on page 15-22](#).

The example output indicates that four simple large objects are stored as part of the table **sriram.catalog**. Two objects are stored in the blobspace **blobPIC** in 6144-byte blobpages. Two more objects are stored in the blobspace **bspc1** in 2048-byte blobpages.

The summary information that appears at the top of the display, **Total pages used by table** is a simple total of the blobpages needed to store simple large objects. The total says nothing about the size of the blobpages used, the number of simple large objects stored, or the total number of bytes stored.

The efficiency information displayed under the **Percent Full** heading is imprecise, but it can alert an administrator to trends in the storage of TEXT and BYTE data.

The following sections use the output shown in [Figure 15-2](#) to demonstrate the idea of average fullness.

Interpreting Blobpage Average Fullness

The first simple large object listed in [Figure 15-2 on page 15-35](#) is stored in the blobspace **blobPIC** and requires one 6144-byte blobpage. The blobpage is 51 to 75 percent full, meaning that the size is between $0.51 * 6144 = 3133$ bytes and $0.75 * 6144 = 4608$. The maximum size of this simple large object must be less than or equal to 75 percent of 6144 bytes, or 4608 bytes.

The second object listed under blobspace **blobPIC** requires two 6144-byte blobpages for storage, or a total of 12,288 bytes. The average fullness of all allocated blobpages is 51 to 75 percent. Therefore, the minimum size of the object must be greater than 50 percent of 12,288 bytes, or 6144 bytes. The maximum size of the simple large object must be less than or equal to 75 percent of 12,288 bytes, or 9216 bytes. The average fullness does not mean that each page is 51 to 75 percent full. A calculation would yield 51 to 75 percent average fullness for two blobpages where the first blobpage is 100 percent full and the second blobpage is 2 to 50 percent full.

Now consider the two simple large objects in blobspace **bspc1**. These two objects appear to be nearly the same size. Both objects require two 2048-byte blobpages, and the average fullness for each is 76 to 100 percent. The minimum size for these simple large objects must be greater than 75 percent of the allocated blobpages, or 3072 bytes. The maximum size for each object is slightly less than 4096 bytes (allowing for overhead).

Applying Efficiency Criteria to Output

Looking at the efficiency information for blobspace **bspc1**, a database server administrator might decide that a better storage strategy for TEXT and BYTE data would be to double the blobpage size from 2048 bytes to 4096 bytes. (Blobpage size is always a multiple of the database server page size.) If the database server administrator made this change, the measure of page fullness would remain the same, but the number of locks needed during an update of a simple large object would be reduced by half.

The efficiency information for blobspace **blobPIC** reveals no obvious suggestion for improvement. The two simple large objects in **blobPIC** differ considerably in size, and there is no optimal storage strategy. In general, simple large objects of similar size can be stored more efficiently than simple large objects of different sizes.

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 14-35.

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 [Virtual-Table Interface Programmer’s Manual](#).

To create an extspace with **onspaces**, use the **-c** option as illustrated in the following example. This example creates a 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 “[Virtual-Table Interface Programmer’s Manual](#).”

For reference information on creating an extspace with **onspaces**, see the utilities chapter in the *Administrator’s Reference*.

Dropping a Extspace

To drop a extspace with **onspaces**, use the **-d** option as illustrated in the following examples. A extspace cannot be dropped if it is associated with an existing table or index.

This example drops a extspace called **pass_space**.

```
% onspaces -d pass_space
```

Skiping 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 dspace1, dspace5
```

This statement causes the database server to skip **dspace1** or **dspace5** 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 **dspace1** and **dspace5** 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 on-line. 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 15-53.](#))

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 15-46.](#))

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 15-46.](#)

Displaying Databases

You can display databases that you create with the following tools:

- SMI tables
- ON-Monitor ♦

UNIX

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*.

UNIX

Using ON-Monitor

To find out the status of a database from within ON-Monitor, select the Status menu, Databases option. [Figure 15-3](#) shows the output of this option.

Database Name	Owner	In Dbspace	When Created	Log Status
sysmaster	informix	rootdbs	07/28/99	U
mydemo	chrisw	rootdbs	09:08:10	U

Figure 15-3
*Output from the
Databases Option of
the ON-Monitor
Status Menu*

When the databases screen appears, ON-Monitor displays the current status of each database. 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 the Database Server for Disabling I/O Errors

The database server notifies you about disabling I/O errors in two ways: the message log and 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 ONDBSPDOWN, as described in the following table.

ONDBSPDOWN Setting	Result	Action
0	Dbospace {space-name} is disabled.	Restore dbospace {space-name}.
	Blobspace {space-name} is disabled.	Restore blobspace {space-name}.
1	The database server must abort.	Reinitialize shared memory.
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 dbospace 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	Dbospace is disabled: 'dbospace-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*.

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.

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about chunks.

onstat -d

The **onstat -d** utility lists all dbspaces and blobspaces 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 approximate number of free blobpages
- The pathname of the physical device

The dbspace flags indicate whether a dbspace (or blobspace) is mirrored. The chunk flags provide the following information:

- Whether the chunk is the primary chunk or the mirrored chunk
- Whether the chunk is on-line, 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.

Sample output for **onstat -D**, which displays the same information plus two additional fields, appears in [Figure 15-4 on page 15-47](#). For descriptions of the **onstat -d** flags, see the utilities chapter in the *Administrator's Reference*.

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 15-4](#) shows sample output.

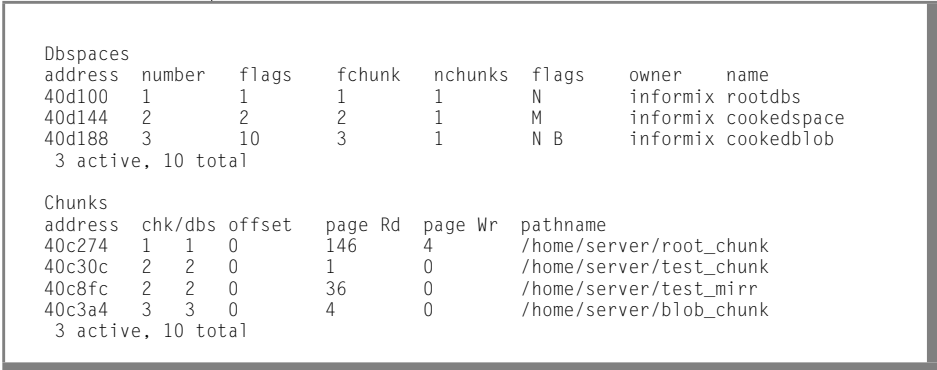


Figure 15-4
onstat -D Output

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 15-5](#) 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 15-5
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 15-6](#) 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 Informix database server reserved pages - PAGE_1DBSP & PAGE_2DBSP
Using dbspace page PAGE_2DBSP.

DBSpace number      1
Flags               1          No mirror chunks
First chunk         1
Number of chunks     2
Date/Time created   07/28/99 14:46:55
DBSpace name        rootdbs
DBSpace owner       informix

Validating Informix Database Server reserved pages - PAGE_1PCHUNK & PAGE_2PCHUNK
Using primary chunk page PAGE_2PCHUNK.

Chunk number        1
Next chunk in DBSpace 2
Chunk offset        0
Chunk size          4000
Number of free pages 1421
DBSpace number      1
Overhead            0
Flags               40          Chunk is online
Chunk name length   23
Chunk path          /home/server/root_chunk
:
:
```

Figure 15-6
oncheck -pr Output
Showing Dbspace
and Chunk
Information

oncheck -pe

To obtain the physical layout of information in the chunk, execute **oncheck -pe**. [Figure 15-7 on page 15-50](#) 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 the extent of 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.

Figure 15-7
oncheck -pe Output

DBSpace Usage Report: rootdbs		Owner: informix	Created: 07/28/99	
Chunk: 1	/home/server/root_chunk	Size	Used	Free
		4000	2579	1421
Disk usage for Chunk 1		Start	Length	
-----		-----	-----	
ROOT DSpace RESERVED Pages		0	12	
CHUNK FREE LIST PAGE		12	1	
TBLSPACE TBLSPACE		13	50	
PHYSICAL LOG Pages		63	500	
LOGICAL LOG Pages - Log 1		563	250	
LOGICAL LOG Pages - Log 2		813	250	
LOGICAL LOG Pages - Log 3		1063	250	
LOGICAL LOG Pages - Log 4		1313	250	
LOGICAL LOG Pages - Log 5		1563	250	
DATABASE TBLSPACE		1813	4	
sysmaster:informix.systables		1817	8	
sysmaster:informix.syscolumns		1825	8	
sysmaster:informix.sysindexes		1833	8	
sysmaster:informix.systabauth		1841	8	
sysmaster:informix.syscolauth		1849	8	
sysmaster:informix.sysviews		1857	8	
sysmaster:informix.sysusers		1865	8	
sysmaster:informix.sysdepend		1873	8	
sysmaster:informix.syssynonyms		1881	8	
Chunk: 2	/home/server/raw_chunk	Size	Used	Free
		500	3	497
Disk usage for Chunk 2		Start	Length	
-----		-----	-----	
OTHER RESERVED Pages		0	2	
CHUNK FREE LIST PAGE		2	1	
FREE		3	497	

UNIX

Using ON-Monitor

Select the Status menu, Spaces option. The first screen indicates whether the space is mirrored. [Figure 15-8](#) shows sample output.

Figure 15-8
*First Screen of
ON-Monitor Spaces
Option*

Press ESC to return to the Status Menu.
Use arrow keys to move the cursor.
Press F3 or CTRL-B for chunk information on the highlighted dbspace/BLOBSpace.

DBSPACES/BLOBSPPACES					
Id	Name	Number of Chunks	When Created	Status	
1	rootdbs	1	08/05/99	N	
2	fstspace	1	08/09/99	Y	
3	fstblob		1 08/09/99	N	

The second screen indicates the mirror status of each chunk. [Figure 15-9](#) shows an example. These flags correspond to the flags that **onstat -d** displays. For an explanation of the flag values, see **onstat** in the utilities chapter of the *Administrator's Reference*.

Figure 15-9
*Second Screen of
ON-Monitor Spaces
Option*

Press ESC to return to the Status Menu.
Use arrow keys to move the cursor.

CHUNKS FOR cooked space						
Chunk Id	Chunk Offset	Pages In Chunk	Pages Used	Full Pathname of Chunk	Status	
2	0	2000	357	/home/server/test_chunk	PO-	
2	0	2000	2000	/home/server/test_mirr	MO-	

Dbspace has total of 2000 pages, 1643 of which are free

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
dbsnum	Number of the dbspace
chksize	Total size of the chunk in pages
nfree	Number of pages that are free
is_offline	Whether the chunk is down
is_recovering	Whether the chunk is recovering
mis_offline	Whether the 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 Tbspaces and Extents

Monitor tablespaces 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.

Using Command-Line Utilities

You can use the following command-line utilities to monitor tablespaces and extents.

oncheck -pt

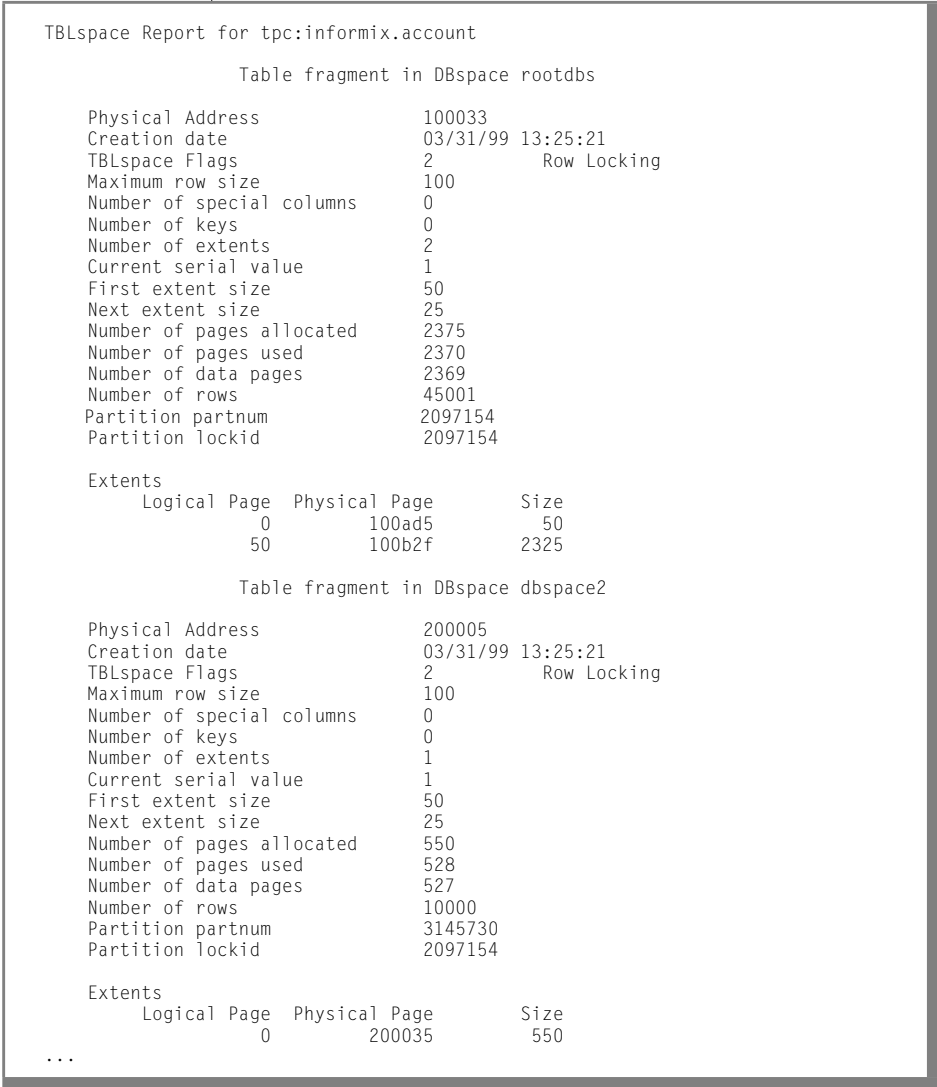
Execute **oncheck -pt** to obtain extent information.

You can include a database-name or table-name parameter with the command. The command displays the following information:

- Number of extents
- Size of the first extent
- Size of the next extent
- Number of pages allocated
- Number of pages used

Figure 15-10 shows sample output. The table in the example is fragmented over multiple dbspaces. Because each fragment of a fragmented table resides in a separate tblspace, the **oncheck -pt** option always displays separate information for each fragment. The number of pages of table data in each fragment is displayed.

Figure 15-10
oncheck -pt Output



oncheck -pT

The **oncheck -pT** option returns all the information from the **oncheck -pt** option as well as the additional information about the index.

Figure 15-11 shows sample output. Each tblspace in the database or table that you supply is listed.

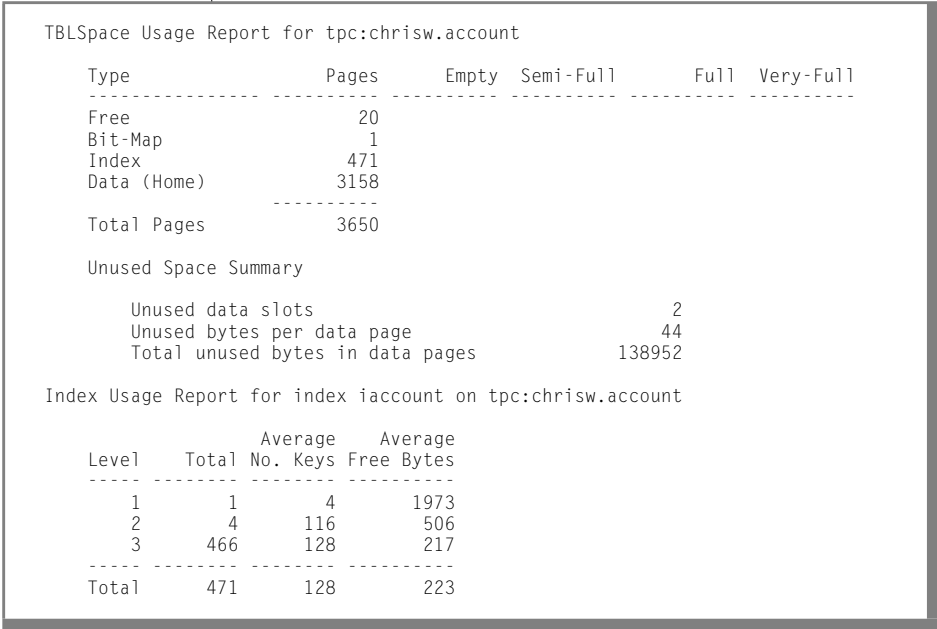


Figure 15-11
Additional
Information shown
by *oncheck -pT*

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.

Using System Catalog Tables

Query the **sysfragments** table to obtain information about all tblspaces that hold a fragment. This table has a row for each tblspace that holds a table fragment or an index fragment. The **sysfragments** table includes the following columns.

Column	Description
fragtype	Table or index fragment
tabid	Table identifier
indexname	Index identifier
partn	Physical location (tblspace ID)
strategy	Distribution scheme (round-robin, expression, table-based index)
dbspace	Dbspacename for fragment
npused	Number of data pages or leaf pages
nrows	Number of rows or unique keys

Not all columns of sysfragments are documented in the preceding list. For a complete listing of columns, see the [Informix Guide to SQL: Reference](#).

Monitoring Simple Large Objects in a BlobSpace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

Using Command-Line Utilities

You can use the following command-line utilities to monitor TEXT and BYTE data in a blobspace.

onstat -d

The **onstat -d** option displays information on each blobspace and the chunks within each blobspace. Figure 15-12 shows an example of the output from this option. In this example, a blobspace called **fstblob** contains a single chunk called **blob_chunk**.

```

Dbspaces
address  number  flags  fchunk  nchunks  flags  owner  name
40c980   1          1      1        1        N    informix rootdb
40c9c4   2          1      2        1        N    informix fstdb
40ca08   3         11      3        1        N B   informix fstblob
3 active, 10 total

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    4000 ~924    1000    POB /home/server/blob_chunk
3 active, 10 total

```

Figure 15-12
*onstat -d Output
with Information
About Blobspaces*

The **onstat -d** option lists the number of free blobpages in each blobspace chunk, as well as the number of total blobpages. The tilde (~) that precedes the `free` value indicates that this number is approximate. The number is approximate because the utility derives it from information stored in the disk version of the chunk free-map page, and not from the version stored in shared memory.

Another complication is that **onstat -d** 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.

onstat -O

The **onstat -O** option displays information about the staging-area blobSpace and the Optical Subsystem memory cache. [Figure 15-13](#) 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**.

```
Subsystem not available

Optical StageBlob Cache
System Cache Totals:
Size  Alloc. Avail.      Number  Kbytes
0      0      0          0      0

System Blob Totals:
Number Kbytes
0      0

User Cache Totals:
SID    User      Size  Number  Kbytes
No sessions currently using the cache

User Blob Totals:
Number Kbytes
```

Figure 15-13
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 blobSpace
kbytes	Size of the simple large objects (in kilobytes) that the database server wrote to the staging-area blobSpace

oncheck -pB

This option gathers its data from the actual blobSpace storage statistics.

Execute **oncheck -pB** with either a database name or a table name as a parameter. The display reports the following statistics:

- Number of blobpages used by this table or database in all blobSpaces
- Blobpage fullness, by blobSpace, for each blobSpace in this table or database

The **oncheck** utility derives the number of free blobpages from the information stored in the shared-memory version of the chunk free-map page, not from the disk version. These statistics are the most current possible and might differ from the output of **onstat -d**, which is derived from the disk version of the free-map page. [Figure 15-14](#) shows sample output.

Figure 15-14
oncheck -pB Output

```
BLOBSpace Report for stores_demo:chrisw.catalog

Total pages used by table          74

BLOBSpace usage:

  Space      Page      Pages  0-25%  26-50%  51-75%  76-100%
  Name      Number
-----
fstblob      0x300000      1      x
fstblob      0x300001      1      x
fstblob      0x300002      1      x
fstblob      0x300003      1      x
  :
fstblob      0x300046      1      x
fstblob      0x300047      1      x
fstblob      0x300048      1      x
fstblob      0x300049      1      x
-----
Page Size is 8192          74
```

oncheck -pe

Execute **oncheck** with the **-pe** options for a detailed listing of chunk use; first the dbspaces are listed, then the blobspaces. The display provides the following blobspace-use information:

- 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

Tip: The **oncheck -pe** option provides information on blobspace use in terms of database server pages, not blobpages.



Figure 15-15 shows sample output.

Figure 15-15
oncheck -pe
Output That Shows
Blobspace Use

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

UNIX

Using ON-Monitor

When you select the Spaces option of the Status menu, the database server displays a series of two screens. The first screen lists any blobspaces.

The second screen lists the following chunk information for each blobspace:

- Chunk ID
- Chunk pathname and offset
- Mirror status flags
- Pages in the chunk
- Number of used disk pages in the chunk

Figure 15-9 on page 15-51 shows an example of this second screen.

Monitoring Simple Large Objects in a Dbspace

You can monitor dbspaces to determine the number of dbspace pages that TEXT and BYTE data use.

To view statistics for simple large objects, execute **oncheck** with the **-pT** options.

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 15-16 shows sample output.

TBLSpace Usage Report for mydemo:chrisw.catalog

Type	Pages	Empty	Semi-Full	Full	Very-Full
Free	7				
Bit-Map	1				
Index	2				
Data (Home)	9				
Data (Remainder)	0	0	0	0	0
Tblspace BLOBs	5	0	0	1	4
Total Pages	24				

Unused Space Summary

Unused data bytes in Home pages	3564
Unused data bytes in Remainder pages	0
Unused bytes in Tblspace Blob pages	1430

Index Usage Report for index 111_16 on mydemo:chrisw.catalog

Level	Total	Average No. Keys	Average Free Bytes
1	1	74	1058
Total	1	74	1058

Index Usage Report for index 111_18 on mydemo:chrisw.catalog

Level	Total	Average No. Keys	Average Free Bytes
1	1	74	984
Total	1	74	984

Figure 15-16
Sample output from
oncheck -pT with
TEXT or BYTE data

Monitoring Sbspaces

For information about monitoring sbspaces, see your [Performance Guide](#).

Features Beyond the Scope of the Database Server

As a Dynamic Server administrator, you need to know the boundaries of Dynamic Server capabilities. This section describes the tasks that lie outside the scope of the database server but are provided by your host computer, operating system, or some other product.

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.

No Compression of TEXT and BYTE Data Types

The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been scanned into a database.

The database server scans TEXT and BYTE data into an existing table in the following ways.

Method to Scan TEXT or BYTE Data	Reference
DB-Access LOAD statement	LOAD statement in the Informix Guide to SQL: Syntax
dbload utility	Informix Migration Guide
Informix ESQL / C programs	Informix ESQL/C Programmer's Manual
High-Performance Loader (HPL)	Guide to the High-Performance Loader

Table Fragmentation and PDQ

In This Chapter	16-3
Fragmentation	16-3
Fragmentation Goals	16-5
Responsibility for Fragmentation	16-6
Fragmentation Strategies	16-6
Table Fragmentation.	16-8
Temporary Table Fragmentation	16-9
Table Index Fragmentation	16-9
SQL Statements That Perform Fragmentation Tasks	16-10
Parallel Database Query	16-11
Parallelism	16-12
Structure of a PDQ Query	16-13
SQL Operators.	16-14
Exchanges	16-15
PDQ Threads	16-16
Use of PDQ	16-16
OLTP Applications	16-17
Decision-Support Applications	16-18
Database Server Use of PDQ	16-19
Fragmentation Enhancement to PDQ	16-20
Memory Grant Manager	16-21
Resource Allocation with PDQ.	16-21
Parameters for Controlling PDQ.	16-21
PDQ Priority	16-22

In This Chapter

This chapter provides an overview of the table fragmentation and parallel database query (PDQ) features of your database server.

Table fragmentation allows you to store the parts of a table on different disks. Table fragmentation allows you to store large amounts of data in a single table and to balance the workload of large queries and high-transaction volumes across multiple disks.

Parallel database query (PDQ) is an Informix database server feature that can improve performance dramatically when the database server processes queries initiated by decision-support applications. PDQ features allow the database server to distribute the work for one aspect of a query among several processors. For example, if a query requires an aggregation, the database server can distribute the work for the aggregation among several processors. PDQ also includes tools for memory-resource management.

PDQ delivers maximum performance benefits when the data that is being queried is in fragmented tables. For information on how to use PDQ and fragmentation for maximum performance, refer to your [Performance Guide](#).

Fragmentation

Fragmentation is a database server feature that enables you to define groups of rows or index keys within a table according to some algorithm or *scheme*. You can store each group or *fragment* in a separate dbspace that is associated with a specific physical disk. You create the fragments and assign them to dbspaces with SQL statements.

From the perspective of an end user or client application, a fragmented table is identical to a nonfragmented table. Client applications do not require any modifications to allow them to access the data that is contained in fragmented tables.

The database server stores the location of each table and index fragment, along with other related information, in the system catalog table named **sysfragments**. You can use this table to access information about your fragmented tables and indexes. For the complete listing of the information in this system catalog table, refer to the [Informix Guide to SQL: Reference](#).

Because the database server has information on which fragments contain which data, the database server can route client requests for data to the appropriate fragment without accessing irrelevant fragments, as [Figure 16-1](#) illustrates. For more information on fragment elimination, refer to your [Performance Guide](#).

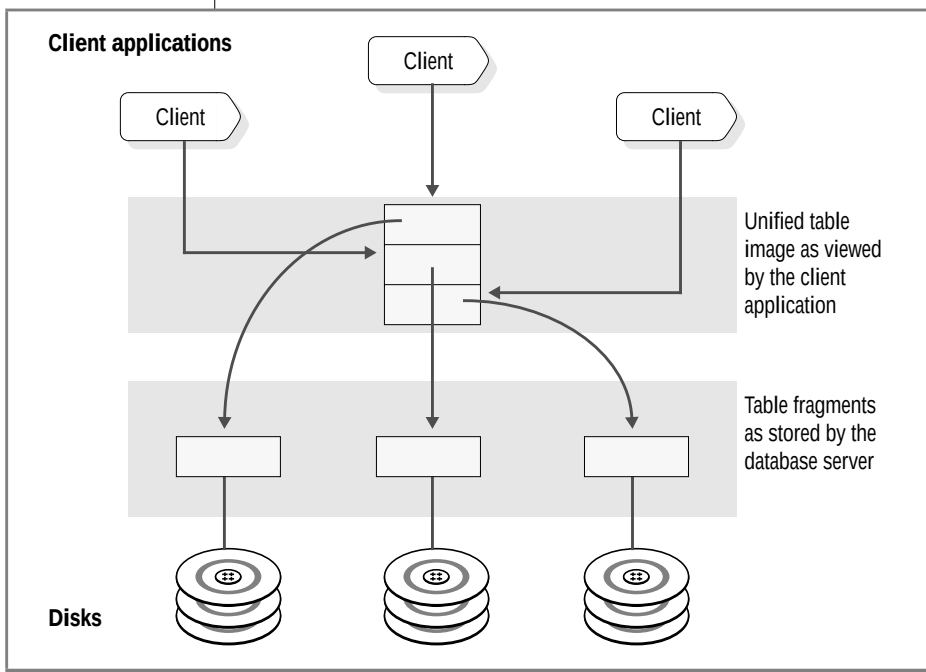


Figure 16-1
*Routing Client
Requests To The
Appropriate Table
Fragments*

The following sections cover these fragmentation topics:

- Fragmentation goals
- Fragmentation strategies
- Summary of SQL statements for fragmentation

Fragmentation Goals

Consider fragmenting your tables if you have at least one of the following goals:

- Improved single-user response time
To improve the performance of individual queries, use fragmentation with parallel database query (PDQ) to scan in parallel fragments that are spread across multiple disks.
- Improved concurrency
Fragmentation can reduce contention for data that is located in large tables that are used by multiple queries and OLTP applications. Fragmentation reduces contention because each fragment resides on a separate I/O device, and the database server directs queries to the appropriate fragment.
- Improved availability
If a fragment becomes unavailable, the database server can still access the remaining fragments.
- Improved ALTER FRAGMENT performance
You can also use the ALTER FRAGMENT TABLE with the ATTACH clause to add data quickly to a very large table.
- Improved backup-and-restore characteristics
Fragmentation gives you a finer backup-and-restore granularity. This granularity can reduce the time that is required for backup-and-restore operations. In addition, you can improve the performance of backup-and-restore operations if you use ON-Bar to perform these operations in parallel.
ON-Archive also performs backup and restore operations in parallel. ♦

Each of the preceding goals has its own implications for the fragmentation strategy that you ultimately implement. [“Fragmentation Strategies” on page 16-6](#) discusses these issues. Your primary fragmentation goal determines, or at least influences, how you implement your fragmentation strategy.

In deciding whether to use fragmentation to meet any of the preceding goals, keep in mind that fragmentation requires some additional administration and monitoring activity. For more information about fragmentation, refer to your [Performance Guide](#).

Responsibility for Fragmentation

Some overlap exists between the responsibilities of the database server administrator and those of the DBA (database administrator) with respect to fragmentation. The DBA creates the database schema. This schema can include table fragmentation. The database server administrator, on the other hand, lays out the disk space and creates the dbspaces where the fragmented tables reside. Because these responsibilities cannot be performed in isolation, implementing fragmentation requires a cooperative effort between the database server administrator and the DBA.

Fragmentation Strategies

A fragmentation strategy consists of two parts:

- A distribution scheme
The scheme that you use to group rows or index keys into fragments is called the *distribution scheme*. You specify the distribution scheme in the `FRAGMENT BY` clause of the `CREATE TABLE`, `CREATE INDEX`, or `ALTER FRAGMENT` statement.
- The set of dbspaces in which you locate the fragments
You specify the set of dbspaces in the `IN` clause of these SQL statements.

The database server supports the following distribution schemes:

- **Round-robin.** This type of fragmentation places rows one after another in fragments, rotating through the series of fragments to distribute the rows evenly.

For smart large objects, you can specify multiple dbspaces in the PUT clause of the CREATE TABLE or ALTER TABLE statement to distribute smart-large objects in a round-robin distribution scheme so that the number of smart-large objects in each space is approximately equal.

For INSERT statements, the database server uses a hash function on a random number to determine the fragment in which to place the row. For INSERT cursors, the database server places the first row in a random fragment, and the second and subsequent rows are assigned to fragments in sequence. If one of the fragments is full, that fragment is skipped.

- **Expression-based.** This type of fragmentation puts rows into fragments based on a *fragmentation expression* that you specify. This expression defines criteria, or rules, for assigning a set of rows to each fragment. The expression can take the form of a range or some other arbitrary rule. You can specify a *remainder fragment* that holds all rows that do not match the criteria for any other fragment, although a remainder fragment reduces the efficiency of the expression-based distribution scheme.

Table Fragmentation

Formulating a fragmentation strategy for a table requires you to make the following decisions:

1. Decide what your primary fragmentation goal is.
Your fragmentation goals depend, to a large extent, on the types of applications that access the table.
2. Decide how the table should be fragmented.
You must make the following decisions:
 - Whether to fragment the table data, the table index, or both
This decision is usually based on your primary fragmentation goal.
 - What the ideal distribution of rows or index keys is for the table
This decision is also based on your primary fragmentation goal.
3. Decide on a distribution scheme.
4. To complete the fragmentation strategy, you must decide on the number and location of the fragments.

For more information on the decisions that you must make to formulate a fragmentation strategy, see the [Informix Guide to Database Design and Implementation](#). For information on optimizing the performance of your fragmentation scheme, refer to your [Performance Guide](#).

Temporary Table Fragmentation

Just as you fragment permanent tables, you also can fragment an explicit temporary table across multiple disks.

To create a temporary, fragmented table, use the TEMP keyword of the CREATE TABLE statement .

You can specify what distribution scheme and which dbspaces to use for the temporary table. For more information on the types of temporary tables, refer to [“Temporary Tables” on page 14-46](#).

You can define your own fragmentation strategy for an explicit temporary table, or you can let the database server dynamically determine the fragmentation strategy. For more information, refer to your [Performance Guide](#).

Table Index Fragmentation

You can fragment both table data and table indexes. When you create an index, you can:

- create an *attached index* by omitting the storage specification from the CREATE INDEX statement.

When you do, the attached index takes on the same fragmentation strategy as the table. Each fragment of an attached index resides in the same dbspace as the corresponding table data.

You create an attached index by omitting the FRAGMENT BY and IN clauses from the CREATE INDEX statement.

```
CREATE TABLE tb1 (a int)
  FRAGMENT BY EXPRESSION
    (a >= 0 and a < 5) IN dbspace1,
    (a >= 5 and a < 10) IN dbspace2
...
;
CREATE INDEX idx1 ON tb1(a);
```

- create a *detached index* by including an explicit storage specification in the CREATE INDEX statement.

When you do, the detached index uses its own fragmentation strategy, which can differ from that of the table. A fragment in a detached index can reside in a different dbspace than the corresponding table data.

You cannot use the round-robin distribution scheme for an index. For more information on the CREATE INDEX statement, refer to the [Informix Guide to SQL: Syntax](#).

Fragmenting table data and table indexes can greatly affect performance. For detailed information on fragmenting table data and table indexes, see your [Performance Guide](#).

SQL Statements That Perform Fragmentation Tasks

To perform most fragmentation tasks, you use appropriate SQL statements. [Figure 16-2](#) lists the fragmentation tasks and the SQL statements to accomplish these tasks.

For details on how to accomplish these fragmentation tasks, refer to the [Informix Guide to Database Design and Implementation](#). For the syntax of these SQL statements, refer to the [Informix Guide to SQL: Syntax](#).

Figure 16-2
Fragmentation Tasks and Corresponding SQL Statements

Fragmentation Task	SQL Statements
Creating a new fragmented table	CREATE TABLE statement, FRAGMENT BY clause
Creating a fragmented table from a single nonfragmented table	ALTER FRAGMENT statement, INIT clause
Creating a fragmented table from more than one nonfragmented table	ALTER FRAGMENT statement, ATTACH clause
Modifying distribution scheme for a fragmented table	ALTER FRAGMENT statement, MODIFY clause

(1 of 2)

Fragmentation Task	SQL Statements
Adding a fragment to a table	ALTER FRAGMENT statement, ADD clause
Removing a fragment from a table	ALTER FRAGMENT statement, DROP clause
Reinitializing a fragmentation scheme	ALTER FRAGMENT statement, INIT clause
Converting a fragmented table to a non-fragmented table	ALTER FRAGMENT statement, INIT clause
Creating a fragmented index	CREATE INDEX statement
Adding an explicit rowid column to a fragmented table	ALTER TABLE statement, ADD ROWID clause

(2 of 2)

Parallel Database Query

Parallel database query (PDQ) refers to the techniques that the database server can use to distribute the execution of a single query over several processors.

The database server can also use PDQ for queries that consume large quantities of non-CPU resources, in particular large quantities of memory and many disk scans.

A query that is processed with PDQ techniques is called a *PDQ query*. When the database server processes a PDQ query, it first divides the query into subplans. The database server then allocates the subplans to a number of threads that process the subplans in parallel. Because each subplan represents a smaller amount of processing time when compared to the original query, and because each subplan is processed simultaneously with all other subplans, the database server can drastically reduce the time that is required to process the query. [Figure 16-3](#) illustrates this concept.

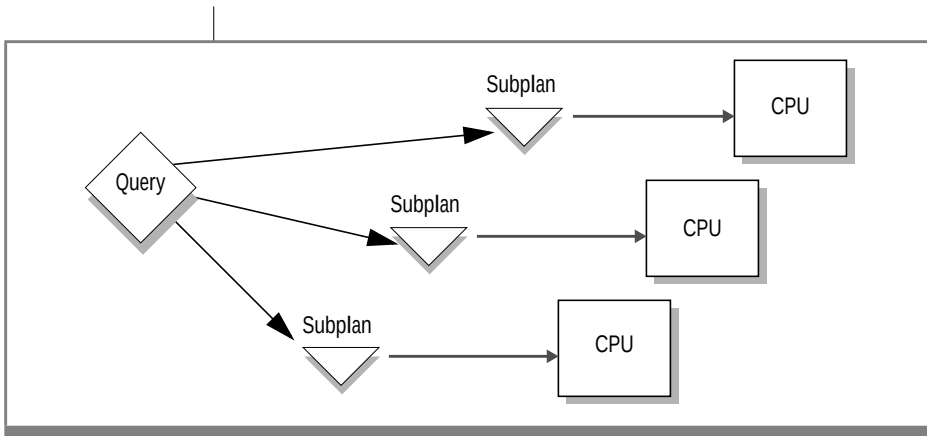


Figure 16-3
Parallel Database Query

Parallelism

The degree of parallelism for a query refers to the number of subplans that the database server executes in parallel to run the query. For example, a two-table join that six threads execute (with each thread executing one sixth of the required processing) has a higher degree of parallelism than one that two threads execute.

PDQ consists of five principal components:

- Parallel scan
- Parallel join
- Parallel sort
- Parallel aggregate
- Parallel group

The database server determines the best degree of parallelism for each component of a parallel database query, based on values set by the database administrator, by the user, and by the client application, as well as various internal considerations such as the number of available virtual processors (VPs), the fragmentation of the tables that are being queried, the complexity of the query, and so on.

At times, the database server uses the components themselves in parallel. For example, consider what occurs when the database server must process a complex join. First, the database server scans the data in parallel. As soon as it has scanned sufficient data to begin the join, it does so. Just after the join begins, the database server begins the sort and performs other required processing until the full join is completed.

You gain the most benefit from PDQ when you use fragmented tables on a multiprocessor computer. However, PDQ can provide performance gains even with nonfragmented tables on a uniprocessor computer.

PDQ provides performance advantages on both uniprocessor computers and multiprocessor computers:

- On a uniprocessor computer, PDQ allows the database server to submit I/O requests to multiple disks in parallel and to take full advantage of the memory on the computer.
- On a multiprocessor computer, PDQ distributes the execution of a query across available processors.

If you want to derive the full performance benefits that PDQ can offer, Informix recommends that you run PDQ on a multiprocessor computer.

Structure of a PDQ Query

The database server divides a query into components that can be performed in parallel to increase the speed of query execution significantly.

Depending on the number of tables or fragments that a query must search, the optimizer determines if a query subplan can execute in parallel.

The Memory Grant Manager (MGM) assigns the different components of a query to different threads across processors.

The **sqlexec** thread initiates these component threads, which the SET EXPLAIN output lists as *secondary threads*.

Secondary threads are further classified as either *producers* or *consumers*, depending on their function. A producer thread supplies data to another thread. For example, a scan thread might read data from shared memory that corresponds to a given table and pass it along to a join thread. In this case, the scan thread is considered a producer, and the join thread is considered a consumer. The join thread, in turn, might pass data along to a sort thread. When it does so, the join thread is considered a producer, and the sort thread is considered a consumer.

The database server uses *SQL operators* and *exchanges* to divide a query plan into subplans that can be performed in parallel to increase the speed of query execution significantly.

SQL Operators

An *SQL operator* is a process that accepts a stream of rows from one or two data tables. Each SQL operator reads each row in a stream and applies a predefined behavior to the data.

For example, a typical query plan might contain scan and hash join SQL operators. The behavior of these SQL operators is as follows:

- **Scan**

This type of SQL operator performs a sequential read on:

- a table or index fragment.
- an unfragmented table or index.

- **Hash join**

This type of SQL uses a hash method to join tables. It selects one table from to constructs a hash table. It then uses that hash table to join data from other tables involved in the join operation.

The database server creates multiple instances of each SQL operator to execute on different parts of the data in parallel, as follows:

- The scan operators execute in parallel, based on the fragmentation strategy of the tables.

The database server structures queries into a plan of SQL operators.

Figure 16-4 shows the SQL operator plan that the database server constructs to process the following SQL query:

```
SELECT geo_id, dollars
FROM customer a, cash b
WHERE a.cust_id=b.cust_id;
```

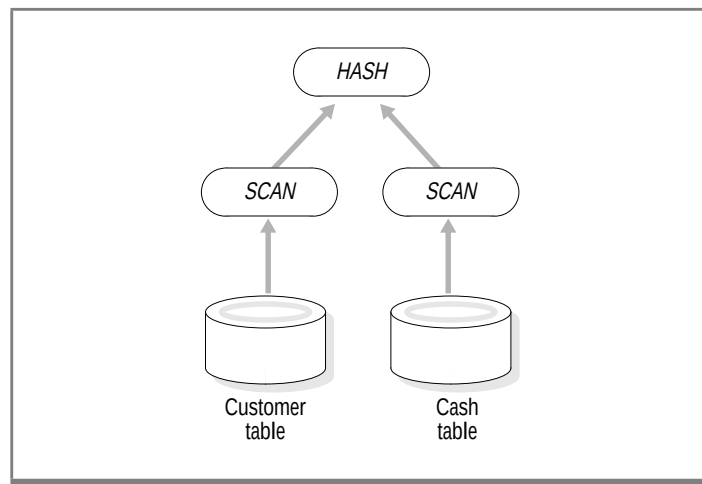


Figure 16-4
SQL Operator Plan
for a Query

Exchanges

An *exchange* is another process that affects parallel processing. An exchange takes the results of two or more instances of an SQL operator and initiates another set of operators to process the next SQL operator that is required to complete the query. The database server inserts exchanges at places within an SQL operator plan where parallelism is beneficial.

When several instances of an SQL operator supply data to another SQL operator, the exchange synchronizes the transfer of data from the multiple instances to the next SQL operator. For instance, if two fragmented tables are to be joined, the optimizer typically calls for a separate scan thread for each

fragment. Because of different I/O characteristics, the scan threads can complete their work at different times. The database server uses an exchange to funnel the data that the various scan threads produce into one or more join threads with a minimum amount of buffering.

PDQ Threads

Depending on the resources that are available for a decision-support query, the database server assigns the different components of a query plan to different threads. The **sqlexec** thread initiates these PDQ threads, which the SET EXPLAIN output lists as *secondary threads*.

The database server creates these secondary threads and exchanges automatically and transparently. They are terminated automatically as they complete processing for a given query. The database server creates new threads and exchanges as needed for subsequent queries.

Use of PDQ

Applications that access data stored in a relational database can be divided into the following two types:

- On-line transaction-processing (OLTP) applications
- Decision-support applications

The complex queries that are typical of decision-support applications can benefit from PDQ.

The next sections describe the characteristics of OLTP and decision-support applications.

OLTP Applications

OLTP applications are characterized by quick, indexed access to a small number of data items. An order-entry system is an example of a typical OLTP system. The transactions handled by OLTP applications are usually simple and predefined.

OLTP applications can be characterized as follows:

- Simple transactions that involve small amounts of data
- Indexed access to data
- Many users
- Frequent requests
- Very fast response times

The default behavior of the database server is ideal for OLTP transactions, optimizing performance for short transactions that require rapid response times. All queries have the same priority for CPU, memory, and disk I/O.

Queries that require quick response and generate only a small amount of information should *not* use PDQ. For example, the following queries should not use PDQ:

- Do we have a hotel room available in Berlin on December 8?
- Does the store in Mill Valley have green tennis shoes in size 4?

The impact of PDQ on OLTP queries can be dramatic. One PDQ parameter limits the number of simultaneous queries the database server can perform. Suppose the number of simultaneous queries is set to 4. If another query requests service, it must wait until one of the previous four queries finishes. If the four queries are decision-support queries, the delay could be several minutes. Typical OLTP queries must be processed immediately.

Decision-Support Applications

Decision-support applications provide information for strategic planning, decision making, and report preparation. Decision-support applications frequently generate queries that require the database server to scan entire tables and manipulate large amounts of data. These queries can require operations such as multiple joins, temporary tables, and hundreds, if not thousands, of calculations. For example, the following queries should use the PDQ features of the database server:

- Based on the predicted number of housing starts, the known age of existing houses, and the observed roofing choices for houses in different areas and price ranges, what roofing materials should we order for each of our regional distribution centers?
- How does the cost of health-care plan X compare with the cost of health-care plan Y, considering the demographic profile of our company? Would plan X be better for some regions and plan Y for others?

Such operations require large amounts of data and large amounts of memory. As a result, the execution times for decision-support applications are far longer than the execution times required for typical OLTP applications. Other typical decision-support applications include payroll, inventory reporting, and end-of-period accounting reports. These applications are frequently executed in a batch environment.

Queries that contain one or more of the following operations require large quantities of memory:

- Hash joins
- Sorting
- Groups

Other factors can also influence how the database server allocates resources to a query. Consider the following SELECT statement:

```
SELECT col1, col2 FROM table1 ORDER BY col1
```

If no indexes exist on **table1**, a sort is required, and hence the database server must allocate memory and temporary disk space to sort the query. However, if column **col1** is indexed, the query does not require these resources.

Decision-support applications have the following characteristics:

- Complex queries that involve large amounts of data
- Large memory requirements
- Few users
- Periodic requests
- Relatively long response times

When both OLTP and decision-support queries are running on the same computer, the database server must balance its resources so that all users receive the best possible performance.

Database Server Use of PDQ

If a user sets the PDQ priority greater than 0 for a given query, the database server treats the query as a parallel database query and controls the PDQ resources allocated to it.

Dynamic Server can execute the following database operations in parallel when PDQ priority is greater than 0 and the involved tables are fragmented into separate dbspaces:

- SELECT statements
- INSERT statements of the form:
 - SELECT...INTO TEMP
 - INSERT INTO...SELECT
- DELETE, and UPDATE statements
- Sorts
- Index builds
- Uncorrelated subqueries
- User-defined routines

For more information about the parallel execution of these operations, see your [Performance Guide](#).

PDQ resources include memory, CPU VPs, disk I/O, and scan threads. The database server determines the degree to which it will parallelize the query based on the following factors:

- The value of PDQPRIORITY
- The availability of computer-system resources (CPUs, memory, and disk I/O)
- The value of parameters (NUMCPUVPS, DS_TOTAL_MEMORY, and so forth) set by the database server administrator

The parameters that control PDQ resources are discussed in the following section. For more information about PDQPRIORITY, see your [Performance Guide](#).



Warning: Users must not set the **PDQPRIORITY** environment variable to a non-zero value for OLTP queries.

Fragmentation Enhancement to PDQ

The complex queries that are typical of DSS applications benefit from PDQ and fragmentation. DSS applications perform complex tasks that often include scans of entire tables, manipulation of large amounts of data, multiple joins, and the creation of temporary tables. Such operations can involve many I/O operations, many calculations, and large amounts of memory.

When you fragment your data across multiple disks, the database server can start parallel SQL operators (scans, sorts, inserts, and so forth) on all available CPU virtual processors. Fragmentation ensures that table fragments are processed in parallel by multiple scan threads.

For more information on fragmentation strategy guidelines to improve performance, refer to your [Performance Guide](#).

Memory Grant Manager

The Memory Grant Manager (MGM) is a Dynamic Server component that coordinates the use of memory, CPU virtual processors (VPs), disk I/O, and scan threads among decision-support queries.

The MGM dynamically allocates the following resources for decision-support queries:

- The number of scan threads started for each decision-support query
- The number of threads that can be started for each query
- The amount of memory in the virtual portion of Dynamic Server shared memory that the query can reserve

For more information about MGM, refer to your [Performance Guide](#).

Resource Allocation with PDQ

This section discusses the configuration parameters, environment variables, and SELECT statements that you can use to balance resource use.

Parameters for Controlling PDQ

[Figure 16-5 on page 16-22](#) summarizes the configuration parameters, environment variables, and the SQL statement that control how the database server allocates resources to PDQ. The value set by the SQL statement supersedes values set by the environment variables, and values set by environment variables supersede values set by configuration parameters.

Figure 16-5
Parameters Used for Controlling PDQ in Dynamic Server

Configuration Parameters	Environment Variables	SQL Statements	Purpose of Parameter
DS_MAX_QUERIES			Maximum number of PDQ queries that can be active at any one time
DS_MAX_SCANS			Maximum number of PDQ scan threads that can execute concurrently
DS_TOTAL_MEMORY			Maximum amount of memory that can be allocated for use by PDQ
MAX_PDQPRIORITY			Percentage of user's requested PDQPRIORITY value that the database server grants
OPTCOMPIND	OPTCOMPIND		Indicate a preferred join type to the query optimizer
	PDQPRIORITY	SET PDQPRIORITY	Request priority and percentage of PDQ resources for an application or a specific query
For more information on the PDQ parameters, refer to the chapter on configuration parameters in the <i>Administrator's Reference</i> and to your Performance Guide. PDQ Priority Users specify what part of database-server resources to devote to PDQ queries by setting the PDQPRIORITY environment variable or using the SET PDQPRIORITY SQL statement. Users can use the SQL statement SET PDQPRIORITY to control the parallelism of individual SQL statements and the PDQPRIORITY environment variable to control the resources allocated to an individual user.			

Users can set PDQPRIORITY to OFF, LOW, or HIGH (values 0, 1, 100, respectively) or integer values between 0 and 100. The values 0 and 1 have special meanings:

- A PDQPRIORITY value of 0 (the default) means that the features of PDQ are not used.
- A PDQPRIORITY value of 1 means to do parallel scans only.
- All other values (2 through 100) represent the percent of the available PDQ resources that the query requests during its execution.

A query running with a PDQ priority of 100 uses all available memory and scan resources. If PDQ priority is less than 100, the query consumes proportionally fewer resources. You can limit the percentage of PDQ resources that the database server actually allocates to users by setting the MAX_PDQPRIORITY configuration parameter.

The environment variable and the SQL statement use the same range of values for setting PDQ priority. In addition, the DEFAULT option for the SET PDQPRIORITY statement allows an application to revert to the value for PDQ priority as set by the environment variable, if any. DEFAULT is the symbolic equivalent of the -1 value.

For more information about the environment variable and the SQL statement, refer to the [Informix Guide to SQL: Reference](#) and the [Informix Guide to SQL: Syntax](#), respectively.



Important: The database server does not decide whether a query should be a parallel database query. If the PDQ priority of a query is greater than zero, the processing of the query is controlled by the PDQ parameters even if the query does not use any parallel processing.

Logging and Log Administration

Chapter 17	Logging
Chapter 18	Managing Database-Logging Status
Chapter 19	Logical Log
Chapter 20	Managing Logical-Log Files
Chapter 21	Physical Logging
Chapter 22	Managing the Physical Log
Chapter 23	Checkpoints and Fast Recovery



Logging

Database Server Processes That Require Logging	17-3
Transaction Logging	17-6
Database Server Activity That Is Logged	17-6
Activity That Is Always Logged	17-7
Activity Logged for Databases with Transaction Logging	17-8
Logging of Simple Large Objects.	17-9
Logging of Smart Large Objects	17-9
Turning On Sbspace Logging.	17-9
Choosing Logging for Smart Large Objects	17-10
Modifying the Logging Status of an Sbspace	17-11
Database-Logging Status.	17-12
Unbuffered Transaction Logging.	17-12
Buffered Transaction Logging.	17-13
ANSI-Compliant Transaction Logging.	17-13
No Database Logging	17-14
Databases with Different Log-Buffering Status	17-14
Transaction Logging and Distributed Queries	17-14
Settings or Changes for Logging Status or Mode.	17-15

In This Chapter

This chapter describes logging of Informix database server functions and addresses the following questions:

- Which database server processes require logging?
- What is transaction logging?
- What database server activity is logged?

Next, the chapter describes logging of databases and addresses the following questions:

- What is the database-logging status?
- Who can set or change the database logging status?

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 off-line (*backed up*). The database server administrator keeps the off-line logical-log records (in 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 19, “Logical Log.”](#)

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 17-6.](#)

- 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 the second phase of 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 23-16.](#))

- Data restoration

During a data restore, you use the logical-log backup with the most-recent storage-space backup to re-create the database server system up to the point of the most recently backed-up logical-log record. After restoring the storage spaces, the database server restores the logical logs to reimplement all the logged activity 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 from the transaction to roll back the transaction. For more information, see SET Database Object Mode in the [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 [Informix Guide to Database Design and Implementation](#). For information on primary key and foreign key constraints, see the [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 30-30.

- 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 High-Availability Data Replication Works](#)” on page 27-8.

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 17-12](#) and [Chapter 18, “Managing Database-Logging Status.”](#)

When you create a database, you specify whether it uses *transaction logging* and, if it does, what log-buffering mechanism it uses. You can change the logging status or buffering mode (change to buffered logging, for example) after the database is created. 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 17-7](#) and [“Transaction Logging and Distributed Queries” on page 17-14](#).

Dynamic Server does not let you turn logging on or off for individual tables.

Database Server Activity That Is Logged

The database server does not generate logical-log records for every operation because it does not need a record of every action. The database server needs logical-log records only to perform the functions listed in [“Database Server Processes That Require Logging” on page 17-3](#). Also, the space required to store a record of everything that the database server did would quickly become unwieldy.

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.

Two 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

The following sections explain the two different types of activity. For more information on the format of logical-log records, see the chapter on interpreting logical-log records in the *Administrator's Reference*.

Activity That Is Always Logged

Some database operations always generate logical-log records, even if you turn off transaction logging for databases on Dynamic Server.

The following operations are always logged for permanent tables:

- SQL data definition statements:

ALTER INDEX	CREATE TABLE
ALTER TABLE	CREATE TRIGGER
CREATE AGGREGATE	CREATE VIEW
CREATE CAST	DROP INDEX
CREATE DATABASE	DROP PROCEDURE
CREATE DATATYPE	DROP SYNONYM
CREATE FUNCTION	DROP TABLE
CREATE INDEX	DROP TRIGGER
CREATE PROCEDURE	DROP VIEW
CREATE ROW	RENAME COLUMN
CREATE SCHEMA	RENAME TABLE
CREATE SYNONYM	
- 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 extents
 - Deallocating extents
 - Truncating
 - Combining chunk free list pages
 - Splitting chunk free list pages
 - Changing the LO header
 - Changing the LO reference count
- Sbspace metadata
- Blobspaces

Activity Logged for Databases with Transaction Logging

If a database uses transaction logging, all SQL data manipulation statements, except SELECT, against that database generate one or more log records. These statements are as follows:

- DELETE
- INSERT
- LOAD
- SELECT INTO TEMP
- UNLOAD
- UPDATE

If these statements are rolled back, the rollback also generates log records.

Logging of 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 large enough to 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.

For more information, see [“Blobspace Logging” on page 19-32](#). For information on how to locate simple-large-object data, see your [Performance Guide](#).

Logging of Smart Large Objects

Sbspaces, described in [“Sbspaces” on page 14-24](#), 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. 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.

Sbspace metadata 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.

Turning On Sbspace Logging

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. You must perform a level-0 backup to capture the initial content of the smart large object.

Figure 17-1 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 [Chapter 23, “Checkpoints and Fast Recovery.”](#)

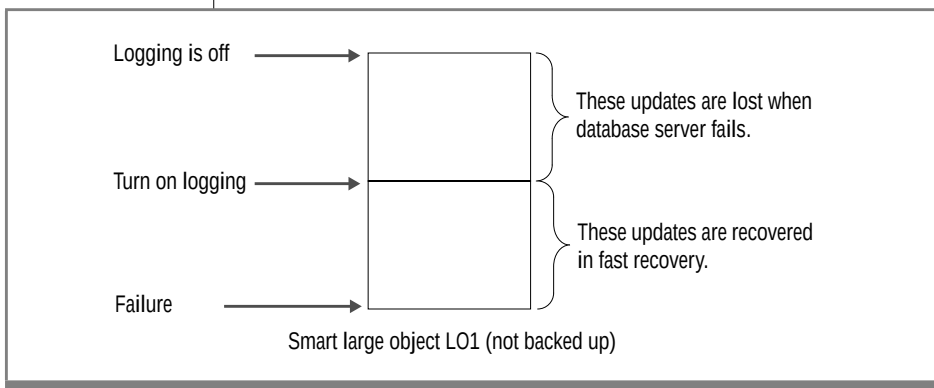


Figure 17-1
*Turning On Logging
in an Sbspace*



Warning: When you turn logging on for a smart large object, you must immediately perform a level-0 backup to ensure that the object is fully recoverable and restorable.

For more information, see [“Backup of Sbspace Data and Log Records” on page 19-33](#) and the *Backup and Restore Guide*.

Choosing Logging for Smart Large Objects

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.

You could load smart large objects with logging turned off and then turn it back on to capture updates to the smart large objects. After you turn logging back on, you must perform a level-0 backup to be able to restore the 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. For information on which smart-large-object operations are logged, see [“Database Server Activity That Is Logged” on page 17-6](#).

Performance is faster when you turn off logging for smart large objects. Turn off logging if users are primarily analyzing the data and updating it infrequently, or if the data is not critical to recover.

Modifying the Logging Status of 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.

Even 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.

You can modify the logging status of an sbspace in any of the following ways:

- Specify the LOG option in the PUT clause of the CREATE TABLE or ALTER TABLE statement. This option turns on logging for all smart large objects that you load into the column. For more information, see [“Altering Storage Characteristics of Smart Large Objects” on page 15-28](#) and the *Informix Guide to SQL: Syntax*.
- Use the **mi_lo_create** DataBlade API function to turn off logging for a smart large object when it is initially loaded. Use the **mi_lo_alter** DataBlade API function to turn on logging after the load is complete. For more information, see the *DataBlade API Programmer's Manual*.
- Use the **ifx_lo_create** ESQ/C function to turn off logging for a smart large object when it is initially loaded. Then use the **ifx_lo_alter** ESQ/C function to turn on logging after the load is complete. For more information, see the *Informix ESQ/C Programmer's Manual*.

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 17-3](#).

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 Transaction Logging” on page 18-11](#). 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 12-6](#) and [“Flushing the Logical-Log Buffer” on page 12-52](#).

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 17-7](#). Usually, you would turn off logging for a database when you are loading data or executing mostly 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.

Transaction Logging and Distributed Queries

If you use distributed queries, the logging status of the databases must be the same (all buffered, all unbuffered, all ANSI compliant, or all without transaction logging). If one of the databases in a distributed query uses transaction logging, the others must also.

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 [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 18, “Managing Database-Logging Status,”](#) 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 [Informix Guide to SQL: Syntax](#).

Managing Database-Logging Status

Changing Database-Logging Status	18-4
Modifying Database-Logging Status with ondblog	18-5
Changing Buffering Status with ondblog	18-5
Canceling a Logging Mode Change with ondblog	18-6
Ending Logging with ondblog	18-6
Making a Database ANSI Compliant with ondblog	18-6
Modifying Database Logging Status with ON-Archive	18-6
Turning On Transaction Logging with ON-Archive	18-7
Canceling a Logging Operation with ON-Archive	18-7
Ending Logging with ON-Archive	18-8
Changing Buffering Status with ON-Archive	18-8
Making a Database ANSI Compliant with ON-Archive	18-8
Modifying Database Logging Status with ontape	18-9
Turning On Transaction Logging with ontape	18-9
Ending Logging with ontape	18-9
Changing Buffering Status with ontape	18-10
Making a Database ANSI Compliant with ontape	18-10
Modifying Database Logging Status with ON-Monitor	18-11
Monitoring Transaction Logging	18-11
Monitoring Transaction Logging with SMI Tables	18-11
Monitoring Transaction Logging with ON-Monitor	18-12

In This Chapter

This chapter covers the following topics on changing the database-logging status:

- Understanding database-logging status
- Modifying database-logging status with **ondblog**
- Modifying database-logging status with ON-Archive
- Modifying database-logging status with **ontape**
- Modifying database-logging status with ON-Monitor
- Monitoring transaction logging

As a database server administrator, you can alter the logging status 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 status, when to use transaction logging, and when to buffer transaction logging, see [Chapter 17, “Logging.”](#) To find out the current logging status of a database, see [“Monitoring Transaction Logging” on page 18-11.](#)

Changing Database-Logging Status

You can use **ondblog**, ON-Archive, **ontape**, or ON-Monitor to add or change logging. Then use ON-Bar, ON-Archive, or **ontape** to back up the data. When you use ON-Bar, **ontape**, or ON-Monitor, the database server must be in on-line or quiescent mode. When you use ON-Archive, the database server must be in on-line mode.

For information on ON-Archive or **ontape**, see the [Archive and Backup Guide](#). For information on ON-Bar, see the [Backup and Restore Guide](#).

[Figure 18-1](#) shows how the database server administrator can change the database-logging status. Certain logging status changes take place immediately while other changes require a level-0 backup.

Figure 18-1
Logging Status 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



Tip: To change the logging status of ANSI-compliant databases, unload and reload the data. For more information, see [“Making a Database ANSI Compliant with ondblog” on page 18-6](#).

Some general points about changing the database-logging status follow:

- When you change the logging status, the database server places an exclusive lock on the database to prevent other users from accessing the database.
- If a failure occurs during a logging-mode change, check the logging mode in ON-Monitor or the flags in the **sysdatabases** table in the **sysmaster** database after you restore the database server data. For more information, see [“Monitoring Transaction Logging” on page 18-11](#).
- 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 [Informix Guide to SQL: Syntax](#).
- Some logging mode changes occur immediately, but 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 Status 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 before the change takes effect. For more information, see the section on using **ondblog** in the *Administrator's Reference*.

Changing Buffering Status with ondblog

To change the buffering status from buffered to unbuffered logging on a database called **stores_demo**, execute the following command:

```
ondblog unbuf stores_demo
```

To change the buffering status 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
```

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

Once you convert a database to ANSI mode, you cannot change it to any other logging mode. To make a database called **stores_demo** into an ANSI-compliant database with **ondblog**, execute the following command:

```
ondblog ansi stores_demo
```

UNIX

Modifying Database Logging Status with ON-Archive

If you use ON-Archive as your backup tool, use the MODIFY/DBLOGGING command to modify database-logging status. See the [Archive and Backup Guide](#).

Before you modify the database-logging status, read “[Changing Database-Logging Status](#)” on page 18-4.

Turning On Transaction Logging with ON-Archive

To add buffered logging to a database called **stores_demo** with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=BUFFERED
```

To add unbuffered logging, use UNBUFFERED as the parameter with the MODE qualifier.

Canceling a Logging Operation with ON-Archive

After you turn on logging for a database with ON-Archive, you can turn logging off again (and unlock the database) before the next level-0 backup of all the storage spaces in the database.

To determine if a database needs to be backed up because logging has been turned on, issue the following query to the **sysmaster** database:

```
SELECT * from sysdatabases
```

If the database needs to be backed up when a logging mode change is pending, the flags field displays a value greater than 255. Query the **syslocks** table to see which locks are in effect.

```
SELECT name FROM sysdatabases WHERE flags > 255
```

If the database has not been backed up, you can cancel the logging mode change with the following command:

```
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=CANCELCHANGE
```

The change takes place immediately; the database is unlocked. You can cancel any number of commands to add transaction logging to a database with the same command.

Ending Logging with ON-Archive

To end logging for a database called **stores_demo** with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=NOLOGGING
```

Changing Buffering Status with ON-Archive

To change the buffering status for a database called **stores_demo** with transaction logging with ON-Archive, use one of the following commands, depending on whether you want the database to have buffered logging:

```
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=BUFFERED
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=UNBUFFERED
```

Making a Database ANSI Compliant with ON-Archive

Once you change the logging status to ANSI compliant, you cannot easily change it again. You must unload and reload the data. For information on the utilities for unloading and loading data, see the [Informix Migration Guide](#).

To make a database called **stores_demo** ANSI compliant with ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores_demo/MODE=ANSI
```


Modifying Database Logging Status with *ontape*

If you use **ontape** as your backup tool, you can use **ontape** to change the logging status of a database. See your [Archive and Backup Guide](#).

Turning On Transaction Logging with *ontape*

Before you modify the database-logging status, read “[Changing Database-Logging Status](#)” on page 18-4.

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. The *ON-Archive* utility allows you to back up specific 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 Status with *ontape*

To change the buffering status 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 status 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*

Once you change the logging status to ANSI compliant, you cannot easily change it again. You must unload and reload the data. For information about how to load and unload data, see the [Informix Migration Guide](#).

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.

UNIX

Modifying Database Logging Status with ON-Monitor

Before you modify the database-logging status, read [“Changing Database-Logging Status” on page 18-4](#).

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 ON-Archive or **ontape**.

To change the log-buffering mode for a database from within ON-Monitor, use the Logical-Logs menu, Databases option.

Monitoring Transaction Logging

This section discusses ways to monitor the logging status of your database and tables. For information on monitoring I/O and memory usage for logging and checkpoints, see your [Performance Guide](#).

Monitoring Transaction Logging with SMI Tables

Query the **sysdatabases** table in the **sysmaster** database to determine the logging status. This table contains a row for each database that the database server manages. The **flags** field indicates the logging status 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 Transaction Logging with ON-Monitor

To use ON-Monitor to find out the logging status of a database, select the Status menu, Databases option. When database logging is on, ON-Monitor shows the buffering mode. [Figure 18-2](#) shows the output of this option.

Database Name	Owner	In Dbspace	When Created	Log Status
sysmaster	informix	rootdbs	07/28/99	U
mydemo	chrisw	rootdbs	09:08:10	U

Figure 18-2
*Output from the
Databases Option of
the ON-Monitor
Status Menu*

When the databases screen appears, ON-Monitor displays the current logging status of each database. 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 Transaction Logging with SMI Tables” on page 18-11](#). The database server uses the following characters to represent database-logging status.

Character	Description
N	No logging
B	Buffered logging
U	Unbuffered logging
A	ANSI compliant

Logical Log

Logical Log	19-3
Logical-Log Files	19-4
Logical-Log Administration	19-5
Size of the Logical Log	19-5
Performance Considerations	19-5
Long-Transaction Considerations	19-6
Smart-Large-Object User Data and Long-Transaction Logging	19-6
Size and Number of Logical-Log Files	19-7
Size of the Logical Log	19-7
Number of Logical-Log Files	19-8
Logging Performance for Blobspaces	19-9
Logical-Log Size When Logging Smart Large Objects	19-9
Location of Logical-Log Files	19-10
Identification of Logical-Log Files	19-11
Status Flags of Logical-Log Files	19-12
Backup of Logical-Log Files	19-14
Logical-Log Restore	19-14
Point-In-Time Restore	19-14
Freeing of Logical-Log Files	19-15
Database Server Attempt to Free a Log File	19-15
Action If the Next Logical-Log File Is Not Free	19-16
Logical Log and Long Transactions	19-17
Factors That Influence the Rate at Which Logical-Log Files Fill	19-18
Factors That Prevent Closure of Transactions	19-20
Setting High-Water Marks	19-20

Logs-Full High-Water Mark	19-22
Emergency Log Backup	19-22
System-Monitoring Interface	19-22
Fast Recovery	19-23
Small Logs, Many Users	19-23
Administrative Activity When Logs Need Backing Up	19-23
Log Space for ON-Archive Tasks	19-24
Logical-Log Administration Tasks for Blobspaces	19-26
Switching Log Files to Activate Blobspaces and Blobspace Chunks	19-26
Backing Up Log Files to Free Blobpages	19-26
Backing Up Blobspaces After Inserting Data	19-27
Backing Up Blobspaces After Deleting TEXT or BYTE Data	19-28
Logging Process	19-29
Dbpace Logging	19-29
Read Page into Shared-Memory Buffer Pool	19-30
Copy the Page Buffer to the Physical-Log Buffer	19-30
Read Data into Buffer and Create Logical-Log Record	19-31
Flush Physical-Log Buffer to the Physical Log	19-31
Flush Page Buffer	19-31
Flush Logical-Log Buffer	19-31
Blobspace Logging.	19-32
Sbpace Logging	19-32
Backup of Sbpace Data and Log Records	19-33
Smart-Large-Object Log Records	19-33

In This Chapter

As the database server administrator, you have responsibilities to configure and manage the logical log. These responsibilities include the following tasks:

- Learning about the logical log
- Allocating an appropriate amount of disk space for the logical log
- Choosing an appropriate location for the logical-log files
- Monitoring the logical-log file status
- Backing up the logical-log files to media

The information in this chapter will help you understand these tasks and the nature of the logging process. For information on how to perform other logical-log tasks, see [Chapter 20, “Managing Logical-Log Files.”](#)

Logical Log

To keep a history of database and database server changes since the time of the last storage-space backup, the database server generates and stores log records. The database server stores the log records in the *logical log*, which is made up of logical-log files. The log is called *logical* because the log records represent units of work related to the 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.

Logical-Log Files

Logical-log files are not files in the operating-system sense of the word *file*. Each logical-log file is a separate allocation of disk space that the database server manages. You must always have at least three logical-log files in the logical log.

The database server administrator needs to be concerned with the logical-log files that make up the logical log. If the files are not managed properly, the database server can suspend processing and, in the worst case, shut down.

The database server administrator must choose an appropriate number, size, and physical location for logical-log files. The following sections discuss these topics:

- [“Size of the Logical Log” on page 19-5](#)
- [“Location of Logical-Log Files” on page 19-10](#)

The database server administrator must also ensure that the next logical-log file is always backed up and free. The following sections discuss this topic:

- [“Identification of Logical-Log Files” on page 19-11](#)
- [“Backup of Logical-Log Files” on page 19-14](#)
- [“Freeing of Logical-Log Files” on page 19-15](#)

Some logical-log file administration tasks relate to managing blobspaces effectively, as discussed in [“Logical-Log Administration Tasks for Blobspaces” on page 19-26](#). Most database users might be concerned with whether transaction logging is buffered.

Logical-Log Administration

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 or whether their transaction logging is buffered. For information on transaction logging, see [Chapter 17, “Logging.”](#) If you want to change the database-logging status, see [“Settings or Changes for Logging Status or Mode” on page 17-15.](#)

Most end users should not be concerned with the logical-log files. The primary administrative tasks include managing individual logical-log files and determining how much disk space to allocate to the logical log.

Size of the Logical Log

In determining how much disk space to allocate, you must balance disk space and performance considerations. 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.

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 logical-log backups and checkpoints, as follows:

- Logical-log backups

When the logical-log files that make up the logical log 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. If enough logical-log disk space is available, however, you can wait for periods of low user activity before you back up the logical-log files. (See [“Backup of Logical-Log Files” on page 19-14.](#))

- Checkpoints

At least one checkpoint record must always be written to the logical log. If you need to free the logical-log file that contains the last checkpoint, the database server must write a new checkpoint record to the current logical-log file. If the frequency with which logical-log files are backed up and freed increases, the frequency at which checkpoints occur increases. Although checkpoints block user processing, they no longer last as long. Because other factors (such as the physical-log size) also determine the checkpoint frequency, this effect might not be significant. (See [“Freeing of Logical-Log Files” on page 19-15.](#))

- Database logging

Whether databases use logging also affects the rate at which the logical log fills.

These performance considerations are related to how fast the logical log fills. The rate at which the logical log fills, in turn, depends on other factors such as the level of user activity on your system. You need to tune the logical-log size, therefore, to find the optimum value for your system.

Long-Transaction Considerations

In addition to the performance considerations discussed in the previous section, you risk a long-transaction situation if logical-log disk space is insufficient. For more information on the long-transaction situation, refer to [“Logical Log and Long Transactions” on page 19-17.](#)

Smart-Large-Object User Data and Long-Transaction Logging

If you choose to log smart-large-object user data, be aware that this option can potentially generate large amounts of logical-log traffic, possibly filling the logical logs. One way to reduce the volume of logical-log entries for fully logged smart large objects is to turn off logging when the smart large object is originally loaded and then reenale logging after the loading is complete. You must perform a level-0 backup after you turn on logging.



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.

Size and Number of Logical-Log Files

After you know how much disk space to allocate for the entire logical log, you can make decisions about how many log files you want and what size.

When you think about the size of the logical-log files, consider these points:

- The minimum size for a logical-log file is 200 kilobytes.
- The maximum size for a logical-log file is essentially unbounded.
- If your tape device is slow, ensure that logical-log files are small enough to be backed up quickly.
- Smaller log files mean slower recovery because you potentially lose the last unbacked-up logical-log file if the disk that contains the logical-log files goes down.
- If you plan to log smart-large-object data, you must ensure that the log size is *considerably* larger than the amount of data being written.

Size of the Logical Log

Use the LOGSIZE configuration parameter to set the size of the logical log. It is difficult to predict how much logical-log space your database server system requires until it is fully in use. The following expression provides the *minimum* total-log-space configuration, in kilobytes, that Informix recommends:

```
LOGSIZE = (users * maxrows) * 512
```

Set *users* to the maximum number of users that you expect to access the database server concurrently. If you set the NETTYPE parameter, you can use the value that you assigned to the NETTYPE **users** field. 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 *users* in the preceding formula.

Set *maxrows* to the maximum number of rows that you expect in the tables.

You can increase the amount of space devoted to the logical log as necessary and in several ways. The easiest way is to add another logical-log file. See [“Adding a Logical-Log File” on page 20-4](#).

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.
- You should create enough logical-log files so that you can switch log files if needed without running out of free logical-log files.
- The number of logical-log files cannot exceed the value of the ONCONFIG parameter LOGSMAX.
- The number of logical-log files affects the frequency of logical-log backups.
- The number of logical-log files affects the rate at which blob space can be reclaimed. See [“Backing Up Log Files to Free Blobpages” on page 19-26](#).

The LOGFILES parameter provides the number of logical-log files. The LOGSIZE parameter provides the size of the logical-log files that are created when the database server initializes disk space. The database server administrator sets both of these configuration parameters in the ONCONFIG file. If all your logical-log files are the same size, you can calculate the total space allocated to the logical-log files as follows:

$$\text{total logical log space} = \text{LOGFILES} * \text{LOGSIZE}$$

If you add logical-log files that are not the size specified by LOGSIZE, 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. For information on how to access the size of logical-log files, see [“Monitoring the Logical Log for Fullness” on page 20-16](#).

For information on LOGSIZE, LOGFILES, and NETTYPE, see the chapter on configuration parameters in the *Administrator’s Reference*.

Logging Performance for Blobspaces

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](#).

Logical-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. Smart-large-object metadata is *always* logged even if the smart large objects are not logged. If you log smart large objects, increase the size of your logical log as the following formula shows:

$$\text{LOGSIZE} = (\text{users} * \text{maxrows}) * 512 + \text{smart\_LOs\_update}$$

Set *smart_LOs_update* to the amount of uncommitted smart large objects that you expect to occur concurrently. The database server logs both a before image and after image of a smart large object for update transactions. If transactions update parts of a large object, account for the before and after image in your estimate of *smart_LOs_update*.

Another way to estimate the log size is to multiply the MIN_EXT_SIZE value by how often you expect to update the smart large objects.

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 20-7](#).

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.

The actual disk space allocated for each logical-log file has an identification number known as the *logid*. For example, if you configure six logical-log files, these files have logid numbers one through six. As logical-log files are backed up and freed, the database server reuses the disk space for the logical-log files. However, the database server continues to increment the unique ID numbers by one. [Figure 19-1](#) illustrates the relationship between the logid numbers and the unique ID numbers.

Figure 19-1
Logical-Log File-Numbering Sequence

Logid Number	First Rotation Unique ID Number	Second Rotation Unique ID Number	Third Rotation Unique ID Number	Fourth Rotation Unique ID Number
1	1	7	13	19
2	2	8	14	20
3	3	9	15	21
4	4	10	16	22
5	5	11	17	23
6	6	12	18	24

For information on how to display the unique ID and logid numbers of a logical-log file, refer to [“Monitoring the Logical Log for Fullness” on page 20-16](#).

Status Flags of Logical-Log Files

All logical-log files have one of the following three status flags in the first position: Added (A), Free (F), or Used (U). Descriptions of all the individual logical-log status flags follow.

Status Flag	Description
Added (A)	A logical-log file has an <i>added</i> status when it is newly added. The logical-log file does not become available for use until you complete a level-0 backup of the root dbspace.
Free (F)	A logical-log file is <i>free</i> when it is 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.
Used (U)	A logical-log file is <i>used</i> when it is still needed by the database server for recovery (rollback of a transaction or finding the last checkpoint record).
Backed-Up (B)	A logical-log file has a <i>backed-up</i> status after it has been backed up.
Current (C)	A logical-log file has a <i>current</i> status if the database server is currently filling the log file.
Last (L)	A logical-log file has a status of <i>last</i> if it contains the most recent checkpoint record in the logical log. This file and subsequent files cannot be freed until the database server writes a new checkpoint record to a different logical-log file.

Figure 19-2 shows the possible log-status flag combinations.

Figure 19-2
Logical-Log Status Flags

Status Flag	Status of Logical-Log File
A-----	Log has been added since the last level-0 storage-space backup. Not available for use.
F-----	Log is free. Available for use.
U	Log has been used but not backed up.
U-B----	Log is backed up but still needed for recovery.
U-B---L	Log is backed up but still needed for recovery. Contains the last checkpoint record.
U---C	Log is the current logical-log file.
U---C-L	Log is the current logical-log file. It contains the last checkpoint record.



Tip: A logical-log file has a status flag of F only if the system has been reinitialized.

To find out the status of a logical-log file, use the methods explained in [“Monitoring the Logical Log for Fullness” on page 20-16](#).

Backup of 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.

You can initiate a manual logical-log backup or set up continuous logical-log backups. If you use ON-Bar to perform logical-log backups, see the *Backup and Restore Guide*. If you use ON-Archive or **ontape** to perform logical-log backups, see the [Archive and Backup Guide](#).

Logical-Log Restore

After you restore the storage spaces, you must restore the logical logs to bring all the data to a consistent state. A restore performed when the database server is on-line is called a warm restore.

Point-In-Time Restore

You can restore storage spaces and logical logs to a particular point in time during a cold restore. If you use ON-Bar to perform point-in-time restores, see the *Backup and Restore Guide*. If you use ON-Archive to perform point-in-time restores, see the [Archive and Backup Guide](#).

Freeing of Logical-Log Files

If you back up a logical-log file, that file is not necessarily free to receive new log records. 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. Perform a full checkpoint using the **onmode -c** command to ensure that the logical-log file does not contain the oldest update.

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.



Database Server Attempt to Free a Log File

The database server attempts to free logical-log files under the following conditions:

- Each time that the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began.
- When the database server first writes to a new logical-log file, it attempts to free the previous log.

The attempt succeeds only if the criteria listed in the preceding section, “Freeing of Logical-Log Files,” are met.

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 23-8](#).

- The file contains an open transaction.

The open transaction is the long transaction discussed in [“Logical Log and Long Transactions” on page 19-17](#). In this situation, you have to recover the database server data from storage-space backups in a full-system restore.

- The file is not backed up.

If the logical-log file is not backed up, processing resumes when you perform the backup. Use ON-Bar or **ontape** to back up the logical-log files.

If you use ON-Archive, you must use the **ondatartr** utility to back up the logical-log files so that the **onarchive** command can access the **sysmaster** database while processing is suspended. For information on **ondatartr**, see the description of emergency log backup in the [Archive and Backup Guide](#). ♦

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.

logid	Logical-Log File Status
1	U-B----
2	U---C--
3	F
4	U-B---L

Logical Log and Long Transactions

A *long transaction* is a transaction that starts in one logical-log file and is not committed when the database server needs to reuse that same logical-log file. In other words, a long transaction spans more than the total space allocated to the logical log.

Because the database server cannot free a logical-log file until all records within the file are associated with closed transactions, the long transaction prevents the first logical-log file from becoming free and available for reuse.

To prevent long transactions from developing, take the following precautions:

- Ensure that the logical-log file does not fill too fast.
- Ensure that transactions do not remain open too long.
- Set high-water marks to have the database server automatically slow down processing when a long transaction is developing.
- Periodically commit writes to smart large objects.

For more information, see [“Smart-Large-Object User Data and Long-Transaction Logging” on page 19-6](#).

The subsequent sections explain these steps.

Factors That Influence the Rate at Which Logical-Log Files Fill

Several factors influence how fast the logical log fills. It is difficult to know exactly which factor is the most important for a given instance of the database server, so you need to use your own judgment to estimate how quickly your logical log fills and how to prevent long-transaction conditions. Consider these factors:

- Size of the logical log
A smaller logical log fills faster than a larger logical log. If you need to make the logical log larger, you can add another logical-log file, as explained in [“Adding a Logical-Log File” on page 20-4](#).
- Number of logical-log records
The more logical-log records written to the logical log, the faster it fills. If databases that your database server manages use transaction logging, transactions against those databases fill the logical log faster than transactions against databases without transaction logging.
- Type of log buffering
As explained in [“Unbuffered Transaction Logging” on page 17-12](#), databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.

- 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. Also, 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.

- 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

The frequency of rollbacks affects the rate at which the logical log fills. More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small. In addition, rollbacks increase the activity in the logical log.

- Number of smart large objects

Because smart-large-object metadata is always logged, a database with a great number of frequently updated smart large objects fills logical logs rapidly.

- Number of smart large objects with user-data logging enabled

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.

Factors That Prevent Closure of Transactions

Several factors influence when transactions close. Be aware of these factors so that you can prevent long-transaction problems:

- **Transaction duration**

The duration of a transaction might be beyond your control. For example, a client that does not write many logical-log records might cause a long transaction if the users permit transactions to remain open for long periods of time. (For example, a user who is running an interactive application might leave a terminal to go to lunch part of the way through a transaction.)

The larger the logical-log space, the longer a transaction can remain open without a long-transaction condition developing. However, a large logical log by itself does not ensure that long transactions do not develop. Application designers should consider the transaction-duration issue, and users should be aware that leaving transactions open can be detrimental.

- **High CPU and logical-log activity**

The amount of CPU activity can affect the ability of the database server to complete the transaction. Repeated writes to the logical-log file increase the amount of CPU time that the database server needs to complete the transaction. Increased logical-log activity can imply increased contention of logical-log locks and latches as well.

Setting High-Water Marks

The database server alters processing at two critical points to manage the long-transaction condition. To tune both points, you can set values in the ONCONFIG file.

The first critical point is the *long-transaction high-water mark*. When the logical log reaches the long-transaction high-water mark, the database server recognizes that a long transaction exists and begins searching for an open transaction in the oldest, used (but not freed) logical-log file. If a long transaction is found, the database server directs the thread to begin to roll back the transaction. More than one transaction can be rolled back if more than one long transaction exists.

The transaction rollback itself generates logical-log records, however, and as other processes continue writing to the logical-log file, the logical log continues to fill.

The second critical point is the *exclusive-access, long-transaction high-water mark*. When the logical log reaches the exclusive-access, long-transaction high-water mark, the database server dramatically reduces log-record generation. Most threads are denied access to the logical log. Only threads that are currently rolling back transactions (including the long transaction) 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.

If the long transactions cannot be rolled back before the logical log fills, the database server shuts down. If this situation occurs, you must perform a data restore. During the data restore, you must *not* roll forward the last logical-log file. Doing so re-creates the problem by filling the logical log again.

The default values for the configuration parameters LTXHWM and LTXEHWL are 50 and 60, respectively. These values eliminate any risk of a long transaction having too little log space in which to roll back. The database server initialization emits a warning if your ONCONFIG file contains values greater than 50 and 60 for these parameters. To overcome these warnings, reduce your parameters to conform. If your log space is finely tuned such that your LTXHWM percentage represents precisely what your longest transaction requires, you will need to add an amount to your log space equal to the difference between your current LTXHWM value and the recommended value of 50.

For information on LTXHWM and LTXEHWL, see the chapter on configuration parameters in the *Administrator's Reference*.

Logs-Full High-Water Mark

To enable the logs-full high-water mark, set the LBU_PRESERVE configuration parameter to 1. When you set LBU_PRESERVE to 1, the database server blocks DB-Access, ESQL/C, and all other clients from generating log records in the last logical-log file when the logs-full condition is reached. The default value of LBU_PRESERVE is 0, or off.

Whenever you change the value of LBU_PRESERVE, you must reinitialize shared memory for the change to take effect.

The LBU_PRESERVE configuration parameter, when set to 1, enables the logical-log high-water mark. The high-water mark prevents transaction records from filling the last free logical-log file. When the high-water mark has passed, client transaction requests are frozen until the logical log is backed up.

Emergency Log Backup

Although the logs-full high-water mark eliminates the need for emergency backup during transaction processing, four known scenarios still require the database server administrators to use emergency logical-log backup. Each case is examined in detail in the following sections.

System-Monitoring Interface

A privileged client is responsible for building the system-monitoring interface (SMI). This client can potentially invade the last logical-log file. If you do not configure sufficient log space or a sufficient number of logical-log files, the privileged client might not succeed in building SMI without a logical-log backup. This situation can cause the logical log to fill.

Fast Recovery

When you start the database server after an uncontrolled shutdown, it needs log space to roll back any transactions that were uncommitted when the shutdown occurred. The threads that perform the recovery have privileges that allow them to use the last logical-log file. Because of this privilege, the logical log might become full, but only in the unlikely case that the number and size of transactions open when the shutdown occurred exceed the size of the logical log.

Small Logs, Many Users

Perhaps you configure your logical log files as follows:

$$\text{Logical Log Size} < 2 * \text{page\_size} * \text{number of users}$$

If all users enter transactions of maximum complexity, applications might invade the last logical-log file with OLTP activity. Only when you set the size of the logical log much smaller than two pages per user can a logs-full condition occur.

Administrative Activity When Logs Need Backing Up

Because certain administrative utilities have the privilege to invade the last logical-log file, you might have to perform an emergency logical-log backup. For example, when the logical log approaches full while you are doing large quantities of administrative work, you might need to perform an emergency logical-log backup.

Dynamic Server uses the following utilities.

onarchive	onautovop	onbar_d
oncatlgr	oncheck	ondatartr
ondblog	onmonitor	onparams
onmsync	onspaces	ontape

UNIX

Log Space for ON-Archive Tasks

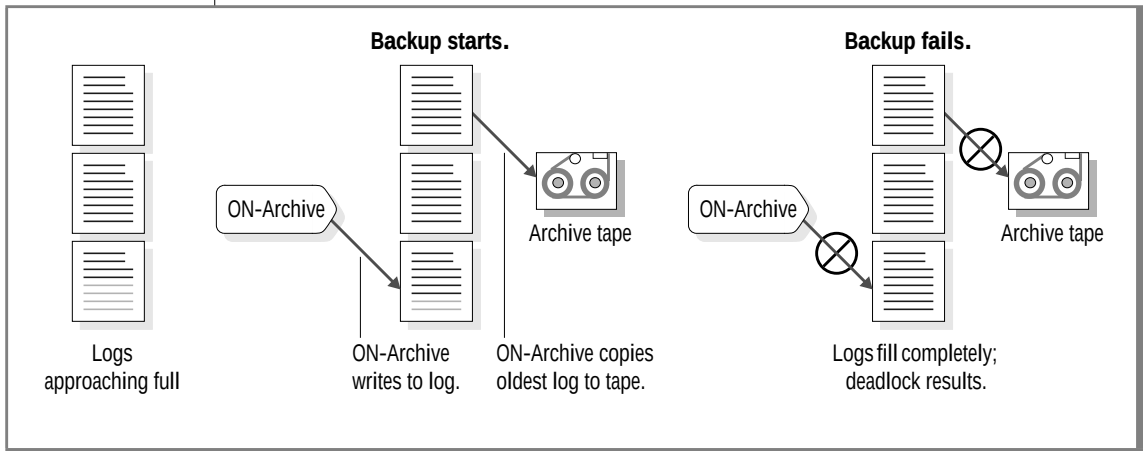
If you use ON-Archive as your backup and restore tool, you might encounter a deadlock during peak activity on high-volume, on-line transaction-processing (OLTP) systems. The deadlock occurs when OLTP activity fills the logical log faster than ON-Archive can back up the logs to tape and free them.

Before marking a backed-up log as free, the database server updates the ON-Archive catalog tables to record the occurrence of the logical-log backup. This update itself generates logical-log activity, leading to a possible deadlock as the logs continue to fill. When a deadlock of this type occurs, the database server administrator must use the emergency log backup procedure even though a continuous logical-log backup is already in progress. For more information on the emergency backup procedure, see the [Archive and Backup Guide](#).

Important: This source of deadlock should never occur for ON-Bar users. ON-Bar marks the logical log as backed up and free, and then it updates the **sysutils** database to record the logical-log backup.

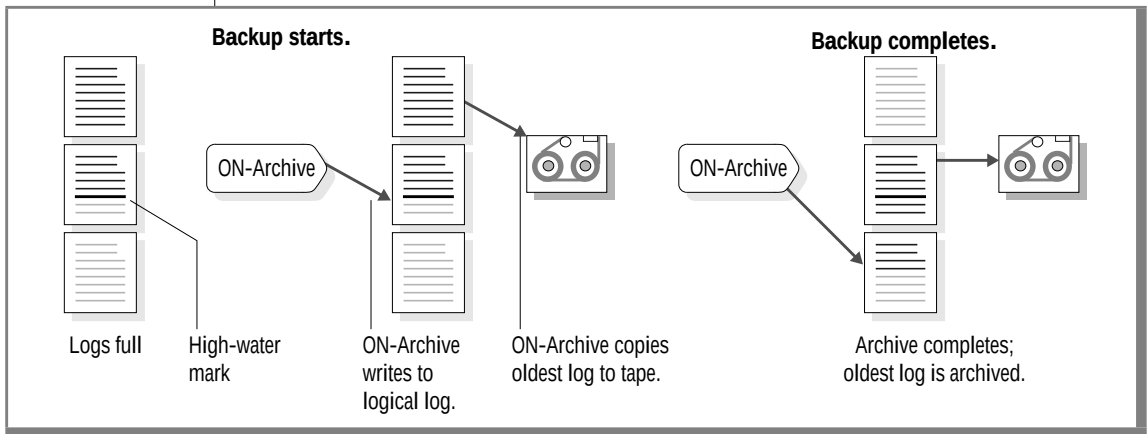
Figure 19-3 illustrates how a deadlock of this type occurs.

Figure 19-3
ON-Archive with High-Water Mark Off



The ON-Archive high-water-mark feature provides a solution for logical-log deadlocks of this type. When you enable this feature, the database server blocks OLTP activity when the next-to-last log fills, rather than the last log, as it usually does. In doing so, the database server preserves the last logical-log file for record logging generated by administrative activities such as a backup of the logical log. [Figure 19-4](#) illustrates how the high-water mark feature prevents logical-log deadlocks.

Figure 19-4
ON-Archive with High-Water Mark On



Logical-Log Administration Tasks for Blobspaces

Applications that use blobspaces require the following logical-log administration tasks. The following sections explain these tasks:

- Switching log files to activate blobspaces
- Switching log files to activate new blobspace chunks
- Backing up log files to free deleted blobspace pages

Switching Log Files to Activate Blobspaces and Blobspace Chunks

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 20-15](#).

Backing Up Log Files to Free Blobspages

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

The following sections explain the reasons for this functionality.

Simple large objects stored in blobspaces require much greater amounts of disk space than other data types. If an application inserts a row that contains TEXT or BYTE data, the database server does not write the actual column data to the logical-log file. The database server writes only the blobpage overhead pages (which include the free-map page and the bit-map page) to the logical log.

The free-map page contains an entry for each blobpage in the blobspace chunk. Each entry contains the following information:

- A flag that indicates whether the page is used or free
- The unique ID of the logical-log file that was current when the page was written to
- The associated tblspace number of the data stored on the page

For information on the blobspace free-map page, see the chapter on disk structures in the *Administrator's Reference*.

Backing Up Blobspaces After Inserting Data

Be sure to back up all blobspaces and logical logs containing transactions on simple large objects stored in a blobspace. Use the following procedure 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. It does not matter whether you back up the logical logs or blobspaces first.

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. 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. This blobspace might not be restorable.

Warning: *If you do not back up these blobspaces and logical logs, you might not be able to restore the blobspace data.*

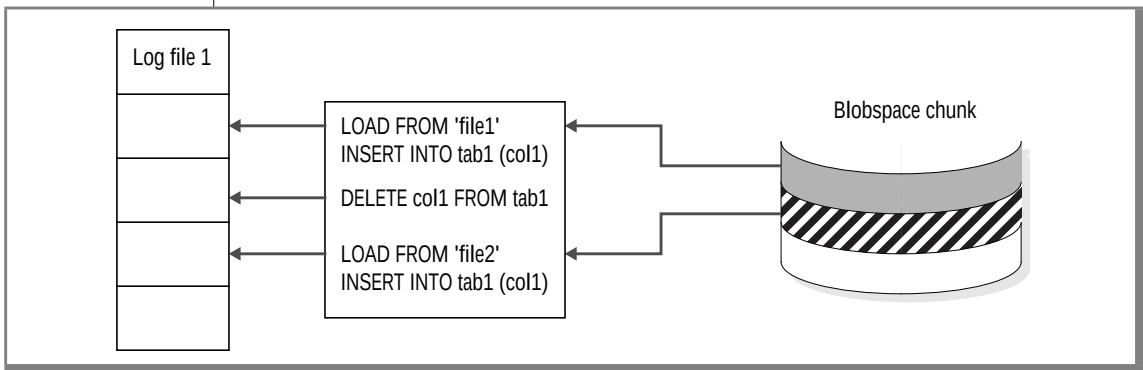


Backing Up Blobspaces After Deleting TEXT or BYTE Data

When an application deletes a row that contains TEXT or BYTE data, the database server marks each blobpage that contained part of the deleted data as FREE in the free-map overhead page. The database server writes a log record to the logical log to record the changes to the free-map page. If the database server has not yet backed up the logical-log file that contains the transaction that inserted the TEXT or BYTE data, the database server does not write over the deleted blobpages on subsequent inserts of TEXT or BYTE data. [Figure 19-5](#) illustrates this scenario.

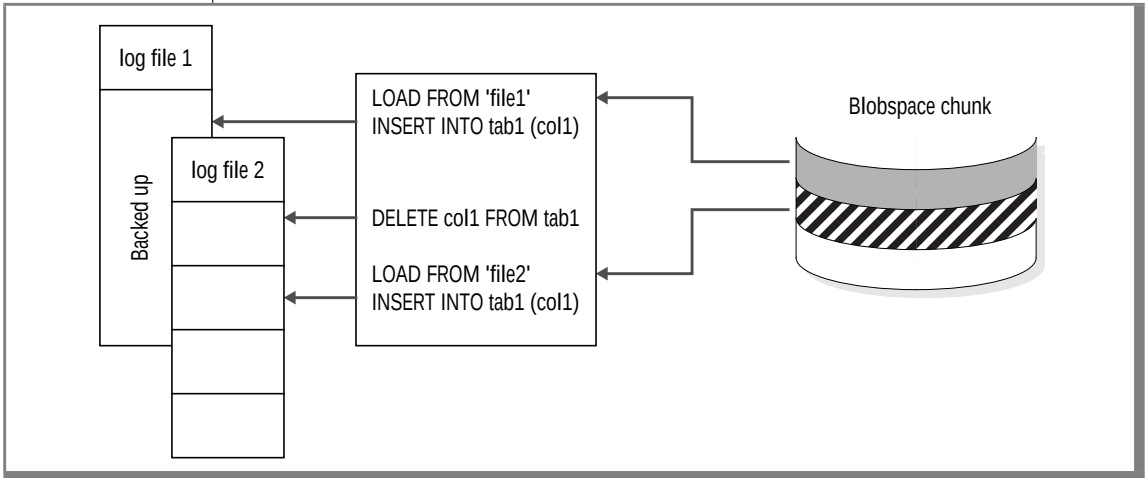
Figure 19-5

No Write to Deleted Blobpages If Logical-Log File Is Not Backed Up



The database server does not write over the deleted blobpages in the scenario that [Figure 19-5](#) shows because it might need to perform a fast recovery and roll forward the transaction that contains the original insert. If the original data is overwritten, the database server has no way of reproducing the transaction.

When you back up the logical-log file that contains the original insert, however, the database server copies the actual TEXT or BYTE data to media. After the data is backed up and available for a restore, the database server can write over the deleted blobpages, as [Figure 19-6](#) shows.

Figure 19-6*Write to Deleted Blobpages If Logical-Log File Is Backed Up*

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. Read the data page from disk to the shared-memory page buffer.
2. Copy the unchanged page to the physical-log buffer, if needed.
3. Write the new data to the page buffer, and create a logical-log record of the transaction, if needed.

4. Flush the physical-log buffer to the physical log on disk.
5. Flush the logical-log buffer to a logical-log file on disk.
6. Flush the page buffer, and write it back to disk.

Read Page into Shared-Memory Buffer Pool

In general, an insert or an update begins when a thread requests a row. The database server identifies the page on which the row resides and attempts to locate the page in the shared-memory buffer pool. If the page is not already in shared memory, the database server reads the page from disk. [“Database Server Thread Access to Buffer Pages” on page 12-42](#) explains this process in more detail.

Copy the Page Buffer to the Physical-Log Buffer

Before the database server modifies a dbospace data page for a nonfuzzy operation, it stores a copy of the unchanged page in the physical-log page buffer if it is needed for fast recovery. The database server eventually flushes the physical-log page buffer that contains this *before-image* to the physical log on disk. Until the database server performs a new checkpoint, subsequent modifications to the same page do not require another before-image to be stored in the physical-log buffer.

The database server knows if a page is already in the physical log. If the time stamp on the page is more recent than the time stamp for the last checkpoint, the page has been changed since the checkpoint and is therefore already in the physical log.

For fuzzy operations, the database server does not store a copy of the before-image of the page into the physical-log page buffer. For fast recovery, the database server finds the oldest update in the logical log. For more information, refer to [“Details of Fast Recovery After A Fuzzy Checkpoint” on page 23-20](#).

Read Data into Buffer and Create Logical-Log Record

The thread that performs the modifications receives data from the application. After the database server stores a copy of the unchanged data page in the physical-log buffer, the thread writes the new data to the page buffer and writes records necessary to roll back or re-create the operation to the logical-log buffer. For more information, refer to [“When the Logical-Log Buffer Becomes Full” on page 12-53](#).

Flush Physical-Log Buffer to the Physical Log

The database server must flush the physical-log buffer before it flushes the data buffer. Flushing the physical-log buffer ensures that a copy of the unchanged page is available until the changed page is written to the physical log. For more information, refer to [“Flushing the Physical-Log Buffer” on page 12-46](#).

Flush Page Buffer

After the database server flushes the physical-log buffer, the database server flushes the data buffer and writes the modified data pages for non-fuzzy operations to disk at the next fuzzy checkpoint. The database server writes all the modified data pages to disk at the next full checkpoint, or when a page cleaner determines that the page should be written to disk. The database server does not flush the data buffer as the transaction is committed. For more information, see [“Flushing Data to Disk” on page 12-45](#).

Flush Logical-Log Buffer

To flush the logical-log buffer, the database server writes the logical-log records to the current logical-log file on disk. For more information, see [“Flushing the Logical-Log Buffer” on page 12-52](#).

For information on the criteria that must be satisfied before the database server frees a logical-log file for reuse, see [“Freeing of Logical-Log Files” on page 19-15](#).

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 bit-map pages) are the only blobspace data that reaches the logical log. By logging these overhead pages, the logical-log file records track blobpage allocation and deallocation (when TEXT or BYTE data is deleted from blobpages), but not the actual TEXT or BYTE data. The logical log records blobspace data only when you back up the logical logs.

Blobspace logging occurs in the following three steps:

1. Blobspace data flows from the network, through temporary buffers in the database server process memory space, and is written directly to disk. If the TEXT or BYTE object requires more than one blobpage, the database server creates links and pointers as needed.
2. If the database uses logging, it writes a record of the operation (insert, update, or delete) to the logical-log buffer. The TEXT or BYTE data is not included in the record (but the information about the location the TEXT or BYTE data is included in the overhead pages).
3. When a logical-log backup begins, the database server uses the logical-log ID number stored in the blobspace free-map page to determine which blobpages to copy to tape. If you use ON-Bar as your backup and restore tool, see the *Backup and Restore Guide* for more details on logical-log backups. If you use ON-Archive as your backup and restore tool, see the [Archive and Backup Guide](#).

Sbspace Logging

When the database server logs smart-large-object data, the database server writes a record of the operation (insert, update, delete, read, or write) to the logical-log buffer. If user-data logging is turned on for the smart large object, the modified portion of the CLOB or BLOB data is included in the log record.

The logical log always records the metadata, which contains pointers to the smart large objects. For more information on sbspace logging, see [“Logging of Smart Large Objects” on page 17-9](#).

Backup of Sbospace Data and Log Records

When you turn on logging for smart large objects, you must back up the sbospace. You can perform logical-log backups and level-0, level-1, and level-2 backups of sbospaces:

- When a level-0 backup of an sbospace begins, the database server uses the chunk-free list to determine which extents are in use. The entire sbospace is backed up.
- When a level-1 or level-2 backup of an sbospace begins, the database server uses the page time stamp and the information in the metadata partition to determine what to back up. Only the sbpages modified since the last backup are backed up.
- When a logical-log backup begins, the database server uses the logical-log ID number stored in the metadata partition to determine which logical logs to back up.

If you use ON-Bar as your backup and restore tool, see the *Backup and Restore Guide* for more details.

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*.

Managing Logical-Log Files

Backing Up Logical-Log Files	20-4
Adding a Logical-Log File	20-4
Adding a Logical-Log File with onparams	20-5
Adding a Logical-Log File with ON-Monitor	20-5
Dropping a Logical-Log File.	20-6
Dropping a Logical-Log File with onparams.	20-6
Dropping a Logical-Log File with ON-Monitor	20-7
Moving a Logical-Log File to Another Dbspace	20-7
Changing the Size of Logical-Log Files	20-9
Using onparams to Add a Logical-Log File with a New Size	20-9
Using a Text Editor to Change the Size of a Log File	20-9
Changing Logical-Log Configuration Parameters	20-10
Using a Text Editor to Change LOGSIZE or LOGFILES	20-10
Using ON-Monitor to Change LOGSIZE or LOGFILES	20-11
Changing LOGSMAX, LTXHWM, or LTXEHWM in the ONCONFIG File	20-12
Changing LOGSMAX, LTXHWM, or LTXEHWM with ON-Monitor	20-12
Freeing a Logical-Log File	20-13
Freeing a Log File with Status A	20-13
Freeing a Log File with Status U	20-13
Freeing a Log File with Status U-B	20-14
Freeing a Log File with Status U-C or U-C-L.	20-14
Freeing a Log File with Status U-B-L	20-15

Switching to the Next Logical-Log File	20-15
Monitoring Logging Activity	20-16
Monitoring the Logical Log for Fullness	20-16
Using Command-Line Utilities	20-17
onstat -l	20-17
oncheck -pr	20-18
Using ON-Monitor.	20-18
Using SMI Tables	20-19
Monitoring Log-Backup Status	20-20
Displaying Logical-Log Records	20-20

In This Chapter

This chapter contains information on managing the database server logical-log files with command line utilities. You must manage logical-log files even if none of your databases uses transaction logging.

The chapter covers the following tasks:

- Adding a logical-log file
- Dropping a logical-log file
- Moving a logical-log file
- Changing the size of a logical-log file
- Changing the logical-log configuration parameters
- Freeing a logical-log file
- Switching to the next logical-log file
- Monitoring log-backup status

For background information regarding the logical log, refer to [Chapter 19, “Logical Log.”](#) When you add or drop logical-log files, the database server must be in quiescent mode.

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 NT.

Backing Up Logical-Log Files

After you add, move, or delete a logical-log file, you must perform a level-0 backup of the root dbspace and the modified dbspace. For instructions on backing up logical-log files with ON-Bar, refer to the *Backup and Restore Guide*.

For instructions on backing up logical-log files with ON-Archive or **ontape**, refer to the [Archive and Backup Guide](#).

Adding a Logical-Log File

You might add a logical-log file for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- As part of moving logical-log files to a different dbspace

You add logical-log files one at a time with the database server in quiescent mode. You cannot add a logical-log file during a storage-space backup.

A newly added logical-log file does not become available until you create a level-0 backup of the root dbspace. This requirement ensures that the backup copy of the reserved pages contains information about the current number of logical-log files.

Verify that you do not exceed the maximum number of logical-log files allowed in your configuration, specified as LOGSMAX.

If you need to, you can increase LOGSMAX. See “[Changing LOGSMAX, LTXHWM, or LTXEHWM in the ONCONFIG File](#)” on page 20-12. Also check the values of LOGFILES and LOGSIZE and increase them if necessary. See “[Using a Text Editor to Change LOGSIZE or LOGFILES](#)” on page 20-10.

You can use either of the following utilities to add a logical-log file:

- **onparams**
- ON-Monitor ♦

Adding a Logical-Log File with onparams

The following command adds a logical-log file to the dbspace called **logspace**, using the log-file size specified by the LOGSIZE configuration parameter:

```
onparams -a -d logspace
```

The status of the new log file is A. The newly added log file becomes available after you create a level-0 backup of the root dbspace and the dbspace that contains the new log file. For information on creating a level-0 backup with ON-Bar, refer to the *Backup and Restore Guide*. For information on creating a level-0 backup with ON-Archive or **ontape**, refer to the [Archive and Backup Guide](#).

For more information on using **onparams** to add a logical-log file, see the utilities chapter in the *Administrator's Reference*.

UNIX

Adding a Logical-Log File with ON-Monitor

If you use ON-Monitor, the size of the new log file is always the value specified by LOGSIZE. Bring the database server to quiescent mode. Select Parameters:Add-Log option to add a logical-log file.

Enter the name of the dbspace where the new logical-log file will reside in the field labelled **Dbspace Name**. The size of the log file automatically appears in the **Logical Log Size** field.

After you add the log file, the status of the new log file is A. The newly added log file becomes available after you create a level-0 backup of the root dbspace and the dbspace that contains the log file. For information on creating a level-0 backup, see [“Backing Up Logical-Log Files” on page 20-4](#).

Dropping a Logical-Log File

You can drop a logical-log file to increase the amount of the disk space available within a dbspace. The database server requires a minimum of three logical-log files at all times.

To make this change, you must log in as either **informix** or **root** on UNIX, and the database server must be in quiescent mode. You must be a member of the **Informix-Admin** group on Windows NT.

Log files that are newly added and have status A do not count toward this minimum of three. You cannot drop a log if your logical log is composed of only three log files.

You drop log files one at a time. You can only drop a log file that has a status of Free (F) or newly Added (A). You must know the logid number of each logical log that you intend to drop.

For information on obtaining a display of the logical-log files and logid numbers, see [“Displaying Logical-Log Records” on page 20-20](#).

Dropping a Logical-Log File with onparams

Execute the following command to drop a logical-log file whose logid number is 21:

```
onparams -d -l 21
```

Create a level-0 backup of the root dbspace and the modified dbspace after you drop the logical-log files. This action ensures that the backup copy of the reserved pages contains information about the current number of logical-log files. This information prevents the database server from attempting to use the dropped log files during a restore. For information on creating a level-0 backup, see [“Backing Up Logical-Log Files” on page 20-4](#).

For information on using **onparams** to drop a logical-log file, see the utilities chapter in the *Administrator's Reference*.

UNIX

Dropping a Logical-Log File with ON-Monitor

Select Parameters:Drop-Log to drop a logical-log file. Use the arrow keys to select the log you want to drop and press CTRL-B or F3. You are asked to confirm your choice.

After you drop the log files, create a level-0 backup of the root dbspace and dbspace that contains the log files. For information on creating a level-0 backup, see [“Backing Up Logical-Log Files” on page 20-4](#).

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 19-10](#). To find out the location of logical-log files, see [“Monitoring Logging Activity” on page 20-16](#).

Changing the location of the logical-log files is actually a combination of two simpler actions:

- Dropping logical-log files from their current dbspace
- Adding the logical-log files to their new dbspace

The database server must be in quiescent mode to move a logical-log file. Although moving the logical-log files is not difficult, it can be time-consuming.

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

1. Free all log files except the current log file.
See [“Freeing a Logical-Log File” on page 20-12.](#)
2. Verify that the value of LOGSMAX is greater than or equal to the number of log files after the move plus 3.
In this case, the value of LOGSMAX must be greater than or equal to 9. Change the value of LOGSMAX, if necessary. See [“Changing LOGSMAX, LTXHWM, or LTXEHWM in the ONCONFIG File” on page 20-12.](#)
3. Drop all but three of the logical-log files.
You cannot drop the current logical-log file. If you have only three logical-log files in the root dbspace, skip this step.
See [“Dropping a Logical-Log File” on page 20-6.](#)
4. Add the new logical-log files to the different dbspace. In this case, add six new logical-log files to **dbspace_1**.
See [“Adding a Logical-Log File” on page 20-4.](#)
5. Create a level-0 backup of the root dbspace and the dbspaces that contain the log files to make the new logical-log files available to the database server. For more information, see [“Backing Up Logical-Log Files” on page 20-4.](#)
6. Switch the logical-log files to start a new current log file.
See [“Switching to the Next Logical-Log File” on page 20-15.](#)
7. Back up the former *current logical-log file* to free it.
8. Drop the three logical-log files that remain in the root dbspace.

Changing the Size of Logical-Log Files

You can change the size of logical-log files in the following ways:

- Use **onparams** to add a new log file of a different size.
This change has no effect on the LOGSIZE parameter. See [“Using onparams to Add a Logical-Log File with a New Size” on page 20-9](#).
- Use a text editor to change the LOGSIZE or LOGFILES parameter in the ONCONFIG file.

Using onparams to Add a Logical-Log File with a New Size

You must use the **onparams** utility to add a new logical-log file with a different size than LOGSIZE. To add a logical-log file with a new size (in this case, 250 kilobytes) to a dbspace called **logspace**, execute the following command:

```
onparams -a -d logspace -s 250
```

Adding a log file of a new size does not change the value of LOGSIZE.

The status of the new log file is A. The newly added log file becomes available after you create a level-0 backup of the root dbspace and the dbspace that contains the log file.

Using a Text Editor to Change the Size of a Log File

Changing LOGSIZE changes the default size for all subsequent logical-log files added but is time-consuming because it requires that you reinitialize the database server to see the change. Subsequent log files are the new size. See [“Using a Text Editor to Change LOGSIZE or LOGFILES” on page 20-10](#).

Changing Logical-Log Configuration Parameters

The following configuration parameters affect the logical-log file and how the database server works with it:

- LOGSIZE
- LOGFILES
- LOGSMAX
- LTXHWM
- LTXEHWM

The following sections explain the procedure for changing each of these configuration parameters. For more information on these parameters, see the chapter on configuration parameters in the *Administrator's Reference*.

You can use a text editor to change these parameters in the ONCONFIG file. You must be logged in as **root** or **informix** on UNIX to change these configuration parameters. You must be a member of the **Informix-Admin** group on Windows NT.

Using a Text Editor to Change LOGSIZE or LOGFILES

If you want to change the size of the log files, you might find it easier to add new log files of the desired size and then drop the old ones.



Important: The changes to *LOGSIZE* and *LOGFILES* do not take effect until you reinitialize the disk space.

To change the size or number of logical-log files

1. Bring the database server off-line or into quiescent mode.
2. To change the size of the log files, change the value of *LOGSIZE* in the ONCONFIG file.
3. To change the number of the log files, change the value of *LOGFILES* in the ONCONFIG file. You might also need to increase the *LOGSMAX* value.

4. Unload all the database server data.
To retain your existing data when you reinitialize the disk, you must unload the data beforehand and reload it once the disk is initialized. This process makes changing these parameters relatively difficult. You cannot rely on storage-space backups to unload and restore the data because a restore returns the parameters to their previous value.
5. Reinitialize disk space.
After the database server disk space is reinitialized, re-create all databases and tables. Then reload all database server data. For more information, see [“Initializing Disk Space” on page 8-4](#).
6. Re-create all databases and tables.
7. Reload all the database server data.
For information on loading and unloading data, see the [Informix Migration Guide](#).
8. Back up the root dbspace to enable your changed logical logs.

UNIX

Using ON-Monitor to Change LOGSIZE or LOGFILES

You can use ON-Monitor to change the value of LOGSIZE or 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 **Initialize** from the **Parameters** menu to reinitialize disk space.
3. Change the value of LOGSIZE or LOGFILES.
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 [Informix Migration Guide](#).

Changing LOGSMAX, LTXHWM, or LTXEHW in the ONCONFIG File

You can use a text editor or ON-Monitor to change the value of LOGSMAX, LTXHWM, or LTXEHW while the database server is on-line. You must be logged in as **root** or **informix** on UNIX to change these configuration parameters. You must be a member of the **Informix-Admin** group on Windows NT.

Changes to these configuration parameters take effect when you shut down and restart the database server. For more information on LOGSMAX, LTXHWM, or LTXEHW, see the chapter on configuration parameters in the *Administrator's Reference*.

UNIX

Changing LOGSMAX, LTXHWM, or LTXEHW with ON-Monitor

You can use ON-Monitor to change LOGSMAX, LTXHWM, or LTXEHW while the database server is on-line.

To change these values

1. Select Parameters:Shared Memory to change one or more of the values. ON-Monitor displays the current values, and you can change the values as follows.

Value to Change	ON-Monitor Field to Change
LOGSMAX	MAX # of Logical Logs
LTXHWM	Long TX HWM
LTXEHW	Long TX HWM Exclusive

2. Shut down and restart the database server for the changes to take effect.

Freeing a Logical-Log File

For a description of what constitutes a free logical-log file, see [“Status Flags of Logical-Log Files” on page 19-12](#).

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 [“Monitoring Logging Activity” on page 20-16](#).



Tip: For information using ON-Bar to back up storage spaces and logical logs, refer to the *“Backup and Restore Guide.”* For information on using ON-Archive or *ontape*, refer to the *“Archive and Backup Guide.”*

Freeing a Log File with Status A

If a log file is newly added (status A), create a level-0 backup of the root dbspace and the dbspace that contains the log file to activate the log file and make it available for use.

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” on page 20-14](#) or [“Freeing a Log File with Status U-B-L” on page 20-15](#).

Freeing a Log File with Status U-B

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 it contains the oldest update which is required for fast recovery.

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 On-Line to Quiescent” on page 8-8](#). 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, currently in use
37 U-C--   This is the current log file, not backed up
```

Freeing a Log File with Status U-C or U-C-L

If you want to free the current log file (status C), follow these steps:

1. Execute the following command:

```
% onmode -l
```

(Be sure to type a lowercase L on the command line, not a number 1.) This command switches the current log file to the next available log file.
2. Back up the original log file with the backup tool that you usually use.

After all full log files are backed up, you are prompted to switch to the next available logical-log file and back up the new current log file. You do not need to do this because you just switched to this log file.

After you follow these steps, if the log file now has status U-B or U-B-L, refer to [“Freeing a Log File with Status U-B” on page 20-14](#) 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

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 on-line 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.)

Monitoring Logging Activity

This section discusses how to monitor the logical-log files. For information on monitoring the logical-log buffers, see [“Monitoring Physical and Logical Logging Activity” on page 22-6](#).

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). This information is important for logical-log management.

Monitoring the Logical Log for Fullness

When the database server is blocking to preserve log space for administrative tasks, the **onstat** utility displays the following message just after its banner line:

```
Blocked: LBU
```

For example, you set LBU_PRESERVE to 1, and every log except the last one is full.

In these circumstances, the second line of any of the **onstat** options appear as shown in the following example:

```
Blocked: LBU
```

To unblock the database server, force a full checkpoint with the **onmode -c** command or a fuzzy checkpoint with the **onmode -c fuzzy** command.

Using Command-Line Utilities

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 third section contains the following information for each logical-log file:

- The address of the logical-log file descriptor
- The logical-log file logid number
- Status flags that indicate the status of each log
 - Flags indicate whether the log is 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

For information on the **onstat -l** option, see the utilities chapter in the *Administrator's Reference*. [Figure 20-1](#) shows sample output.

...							
address	number	flags	uniqid	begin	size	used	%used
846640	1	F-----	0	100233	250	0	0.00
84665c	2	F-----	0	10032d	250	0	0.00
846678	3	U---C-L	3	100427	250	175	70.00
846694	4	F-----	0	100521	250	0	0.00
8466b0	5	F-----	0	10061b	250	0	0.00

Figure 20-1
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 20-2](#) shows sample output.

```
...
Log file number          1
Log file flags           0
Time stamp              6964
Date/Time file filled   07/28/99 14:48:32
Unique identifier        0
Physical location       100233
Log size                 250
Number pages used        0
...
```

Figure 20-2
oncheck -pr Output
Containing Logical-
Log File Information

UNIX

Using ON-Monitor

The Logs option of the Status menu displays much of the same information for logical-log files as the **onstat -l** option displays. In addition, a column contains the dbspace in which each logical-log file is located. [Figure 20-3](#) shows sample output.

```
...
INDIVIDUAL LOG FILES:
Number  Flags    Uniqid  Dbspace    Pages    Used    % Used
  1     F-----    0    rootdbs     250      0     0.00
  2     F-----    0    rootdbs     250      0     0.00
  3     U---C-L    3    rootdbs     250     175    70.00
  4     F-----    0    rootdbs     250      0     0.00
```

Figure 20-3
*Output from the
Logs Option of the
ON-Monitor Status
Menu*

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

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.

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*.

Physical Logging

Physical Logging	21-3
Purpose of Physical Logging	21-4
Fast Recovery Use of Physically-Logged Pages	21-4
Backup Use of Physically-Logged Pages	21-4
Database Server Activity That Is Physically Logged	21-5
Physical Logging and Simple Large Objects	21-5
Physical Logging and Smart Large Objects	21-5
Size and Location of the Physical Log	21-6
Limit to the Size of the Physical Log	21-7
Physical-Log Overflow When Many Users Are in Critical Sections	21-7
Effect of Checkpoints on the Physical-Log Size	21-7
Physical-Log Overflow When Transaction Logging Is Turned Off	21-8
Physical-Log Overflow During Rollback of a Long Transaction .	21-8
Location of the Physical Log	21-9
Details of Physical Logging	21-9
Page Is Read into the Shared-Memory Buffer Pool.	21-10
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer . .	21-10
Change Is Reflected in the Data Buffer	21-10
Physical-Log Buffer Is Flushed to the Physical Log	21-10
Page Buffer Is Flushed	21-11
When a Checkpoint Occurs	21-11
How the Physical Log Is Emptied	21-11

In This Chapter

This chapter defines the terms and explains the concepts that you need to know to perform effectively the tasks described in [Chapter 22, “Managing the Physical Log.”](#) The chapter covers the following topics:

- Physical logging and the purposes it serves
- Some guidelines for the size and location of the physical log
- Details of the physical-logging process

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. Before the database server modifies certain pages in the shared-memory buffer pool, it stores an unmodified copy of the page (called a *before-image*) in the physical-log buffer in shared memory.

The *physical log* is a set of contiguous disk pages where the database server stores before-images.

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. Once a checkpoint occurs, the database server empties the physical log (except in the special circumstances explained in [“Limit to the Size of the Physical Log” on page 21-7](#)). For more information on checkpoints, see [“Checkpoints” on page 23-4](#).



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 Checkpoint” on page 23-5](#).

Purpose of Physical Logging

This seemingly odd activity of storing copies of pages before they are changed ensures that 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. These snapshots are potentially used in two activities: fast recovery and the database server backup.

Fast Recovery Use of Physically-Logged Pages

After a failure, the database server uses the before-images of pages modified by non-fuzzy operations in the physical log 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. [Chapter 23, “Checkpoints and Fast Recovery,”](#) explains this procedure in more detail.

Backup Use of Physically-Logged Pages

When you perform a storage-space backup, the database server performs a full checkpoint and checks disk pages to see which should be backed up. If a backup is active, the database server sends physically-logged pages to the backup program. For more details, see the [Archive and Backup Guide](#) if you use ON-Archive as your backup and restore tool or the *Backup and Restore Guide* if you use ON-Bar.

Database Server Activity That Is Physically Logged

All dbspace page modifications except the following ones are 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 allocated and that are located in a dbspace where no table has been dropped since the last checkpoint.
- Pages for fuzzy operations such as inserts, deletes, and updates.

In case of multiple modifications before the next checkpoint, only one before-image is logged in the physical log (the first before-image).

Storing all before-images of page modifications in the physical log might seem excessive. But 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.

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 tablespaces. 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 of Simple Large Objects” on page 17-9](#).

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 14-24](#).

Size and Location of the Physical Log

When you consider how large to make your physical log, you can begin by using the following formula to calculate an approximate size:

$$\text{PHYSFILE} = (\text{connections} * \text{max\_log\_pages\_per\_crit\_sect} * 4 * \text{pagesize}) / 1024$$

This PHYSFILE value represents a maximum. 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](#).

Variable in Formula	Description
<i>connections</i>	Maximum number of users that you expect to access the database server concurrently. If you set the NETTYPE parameter, connections is the sum of the values specified in the users 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 the following part of the formula represents only 25 percent of the physical log: <i>connections * 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 PHYSFILE parameter in units of kilobytes

Limit to 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 23-5](#).) However, the physical log could still become full, as the following sections describe.

Physical-Log Overflow When Many Users Are in Critical Sections

Under normal processing, once a checkpoint is requested, and the checkpoint begins, all threads are prevented from entering critical sections of code. (See [“Critical Sections” on page 23-4](#).) However, threads *currently* in critical sections can continue processing. The physical log can become full if many threads in critical sections are processing work *and* if the space that remains in the physical log is very small. The many writes that are performed as threads complete their critical section processing could conceivably cause the physical log to become full.

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. Consider the following example about simple large objects in dbspaces stored in a database created with transaction logging.

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.

Physical-Log Overflow During Rollback of a Long Transaction

This same unlikely scenario could occur during the rollback of a long transaction after the second long-transaction high-water mark, LTXEHW, is reached. (See [“Logical Log and Long Transactions” on page 19-17.](#)) After the LTXEHW is reached, and after all threads have exited critical sections, only the thread that is performing the rollback has access to the physical and logical logs. However, the writes that are performed as threads complete their processing could conceivably fill the physical log during the rollback if the following conditions occur simultaneously:

- Many threads were in critical sections.
- The space remaining in the physical log was very small at the time that the LTXEHW was reached.

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 on 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 22-3.](#))

Because the physical log is critical, Informix recommends that you mirror the dbspace that contains the physical 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. If the same page is modified again before the next checkpoint, it does not require another before-image to be stored in the physical-log buffer. (Fuzzy operations do not physically log before-images of pages.) 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 12-23](#).

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 12-46](#).

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 12-45](#).

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*. After a full checkpoint occurs, all modified pages in shared memory are flushed to disk. The database server is physically consistent because all changes to the data since the prior checkpoint are recorded on disk and in the logical log. For information, see [“Full Checkpoint” on page 23-5](#).

After a fuzzy checkpoint occurs, the database server might not be physically consistent, but the checkpoint completes much quicker and does not tie up the database server during heavy update activity. All changes to the data since the last full checkpoint or fast recovery are recorded in the logical log. For information, see [“Fuzzy Checkpoint” on page 23-5](#).

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.

Managing the Physical Log

Changing the Physical-Log Location and Size	22-3
Reasons to Change the Physical-Log Location and Size	22-4
Preparing to Make the Changes	22-4
Checking For Adequate Contiguous Space	22-4
Using a Text Editor to Change Physical-Log Location or Size	22-5
Using onparams to Change Physical-Log Location or Size	22-5
Using ON-Monitor to Change Physical-Log Location or Size	22-6
Monitoring Physical and Logical Logging Activity	22-6
Using Command-Line Utilities	22-7
onstat -l	22-7
oncheck -pr	22-8
Using SMI Tables	22-9
Using ON-Monitor	22-9

In This Chapter

This chapter describes procedures for changing the location and size of the physical log and for monitoring the physical log. For background information about the physical log, see [Chapter 21, “Physical Logging.”](#)

Changing the Physical-Log Location and Size

You can change your physical-log location or size in three ways:

- Using a text editor 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 NT when you make the changes. The following sections describe each of these methods.

For any of the three 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 to reinitialize shared memory. If you use **onparams**, you can reinitialize shared memory in the same step.

Create a complete level-0 backup immediately after you reinitialize shared memory. This storage-space backup is critical for database server recovery.

Reasons to Change the Physical-Log Location and Size

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 [“Location of the Physical Log” on page 21-9](#). For advice on sizing the physical log, see [“Size and Location of the Physical Log” on page 21-6](#). To obtain information about the physical log, see [“Monitoring Physical and Logical Logging Activity” on page 22-6](#).

Preparing to Make the Changes

The space allocated for the physical log must be contiguous. If you move the physical log to a dbspace without adequate contiguous space, or if you increase the log size beyond the available contiguous space, a fatal shared-memory error occurs when you attempt to reinitialize shared memory with the new values. If this error occurs, resize the physical log, or choose another dbspace with adequate contiguous space and then reinitialize shared memory.

Checking For Adequate Contiguous Space

You can check if adequate contiguous space is available with the **-pe** option of the **oncheck** utility. 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 15-46](#).

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 on-line 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 reinitialize shared memory, 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*. For information on using ON-Bar, refer to the *Backup and Restore Guide*. For information on using ON-Archive or **ontape**, refer to the [Archive and Backup Guide](#).

UNIX

Using ON-Monitor to Change Physical-Log Location or Size

Select Parameters:Physical-Log to change the size or dbspace location, or both.

The **Physical-log Size** field displays the current size of the log. Enter the new size (in kilobytes) if you want to change the size of the log. The **Dbspace Name** field displays the current location of the physical log. Enter the name of the new dbspace if you want to change the log location.

You are prompted, first, to confirm the changes.

```
Do you really want to shut down?
```

The second prompt checks if you want to shut the database server down.

```
Do you really want to continue?
```

This last message refers to reinitializing shared memory. If you respond Y, ON-Monitor reinitializes shared memory, and any changes are implemented immediately. The database server displays messages that the database server is shutting down and then initializing and recovering. If you respond N, the values are changed in the configuration file, but the changes do not take effect until you reinitialize shared memory.

After you reinitialize shared memory, create a level-0 backup immediately to ensure that all recovery mechanisms are available.

Monitoring Physical and Logical Logging Activity

This section discusses monitoring the physical-log file, physical-log buffers, and logical-log buffers.

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](#).

Using Command-Line Utilities

You can use the following command-line utilities to obtain information about the physical-log file.

onstat -l

The first line of the **onstat -l** output displays the following information for each physical-log buffer:

- The number of buffer pages used (**bufused**)
- The size of each physical log buffer in pages (**bufsize**)
- The number of pages written to the buffer (**numpages**)
- The number of writes from the buffer to disk (**numwrits**)
- The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)

The second line of the **onstat -l** output displays the following information about the physical log:

- The page number of the first page in the physical-log file (**phybegin**)
- The size of the physical-log file (**physize**)
- The current position in the log where the next write occurs (**physpos**)
- The number of pages in the log that have been used (**phyused**)
- The percentage of the total physical-log pages that have been used (**%used**)

The third line of the **onstat -l** output displays the following information about each logical-log buffer:

- The number of buffer pages used (**bufused**)
- The size of each logical-log buffer in pages (**bufsize**)
- The number of records written to the buffer (**numrecs**)

- The number of pages written to the buffer (**numpages**)
- The number of writes from the buffer to disk (**numwrits**)
- The ratio of records to pages in the buffer (**recs/pages**)
- The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)

Figure 22-1 shows sample output from the **onstat -l** option that contains the relevant fields.

```
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 22-1
onstat -l Output That Shows Information About the Physical and Logical Logs

oncheck -pr

The database server stores the physical-log file information in those reserved pages dedicated to checkpoint information (PAGE_1CKPT and PAGE_2CKPT). You can view the checkpoint reserve pages with the **oncheck -pr** command. The reserve pages contain the state of the physical log at the last checkpoint. Figure 22-2 shows an example of the relevant output.

```
Validating Informix database server reserved pages - PAGE_1CKPT & PAGE_2CKPT
Using check point page PAGE_2CKPT.

Time stamp of checkpoint      16024
Time of checkpoint            07/30/99 09:34:33
Physical log begin address    10003f
Physical log size              500
Physical log position at Ckpt  e9
....
```

Figure 22-2
oncheck -pr Output That Includes Physical-Log Information

Using SMI Tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The following rows contain the relevant 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
llgreys	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

UNIX

Using ON-Monitor

All the information that the **onstat -l** utility provides on the physical-log and logical-log buffers is also available if you select **Logs** from the **Status** menu. [Figure 22-3](#) shows an example of the output.

```

PHYSICAL LOG:
Buffer  Bufsize  Bufused  Numpages  Numwrites  pages/10
P-1      16         0         0         0         0.00
        Phybegin  Physize   Phypos    Phyused   % Used
        10003f    500      236      0         0.00

LOGICAL LOG:
Buffer  Bufsize  Bufused  Numrecs  Numpages  Numwrites  Recs/Page  Pages/10
L-2      16         0         1         1         1         1.00      1.00

```

Figure 22-3
Output from the
Logs Option of the
ON-Monitor Status
Menu

Checkpoints and Fast Recovery

How the Database Server Achieves Data Consistency	23-3
Critical Sections	23-4
Checkpoints	23-4
Full Checkpoint	23-5
Fuzzy Checkpoint	23-5
Fuzzy Operations	23-5
Write-Ahead Logging and Fast Recovery	23-6
Fuzzy Checkpoints Improve Performance	23-6
Events That Initiate a Fuzzy Checkpoint	23-7
Events That Initiate a Full Checkpoint	23-7
Forcing a Full Checkpoint	23-8
Forcing a Fuzzy Checkpoint	23-9
Sequence of Events in a Checkpoint.	23-9
User Threads Cannot Enter a Critical Section	23-10
Logical-Log Buffer Is Flushed to the Logical-Log File on Disk	23-10
Page-Cleaner Thread Flushes the Physical-Log Buffer	23-10
Page-Cleaner Threads Flush Modified Pages in the Buffer Pool	23-10
Checkpoint Thread Writes Checkpoint Record.	23-11
Physical Log Is Logically Emptied	23-11
Backup and Restore Considerations.	23-11
Time Stamps	23-12
Time Stamps on Disk Pages	23-12
Time Stamps on Logical-Log Pages	23-12
Time Stamps on Blobpages.	23-13
Blobpage Time Stamps with Dirty Read Isolation Level	23-13
Blobpage Time Stamps with Committed Read Isolation Level	23-14

Fast Recovery	23-14
Need for Fast Recovery	23-15
Situations When Fast Recovery Is Initiated	23-15
Fast Recovery and Buffered Logging	23-16
Fast Recovery and No Logging	23-16
Details of Fast Recovery After A Full Checkpoint	23-16
Returning to the Last-Checkpoint State	23-17
Finding the Checkpoint Record in the Logical Log	23-18
Rolling Forward Logical-Log Records.	23-18
Rolling Back Incomplete Transactions.	23-18
Details of Fast Recovery After A Fuzzy Checkpoint	23-20
Returning to the Last-Checkpoint State for Nonfuzzy Operations	23-21
Locating the Oldest Update in the Logical Log.	23-21
Applying the Log Records for Fuzzy Operations	23-22
Rolling Forward Logical-Log Records.	23-23
Rolling Back Incomplete Transactions.	23-23
Monitoring Checkpoint Information	23-24
Using onstat Options	23-24
Using ON-Monitor.	23-25
Profile Option	23-25
Force-Ckpt Option	23-25
Using SMI Tables	23-26

In This Chapter

This chapter describes how the database server achieves data consistency through checkpoints and the fast-recovery feature. Read this chapter if you are interested in learning how checkpoints and fast recovery work.

This chapter covers the following topics:

- How the database server achieves data consistency
- Critical sections
- Checkpoints to achieve data consistency
- Time stamps to synchronize events
- Fast recovery
- Monitoring checkpoints

How the Database Server Achieves Data Consistency

The database server uses the following three procedures to ensure that data destined for disk is actually recorded intact on disk:

- Critical sections
- Checkpoints
- Time stamps

These procedures ensure that multiple, logically related writes are recorded as a unit; that data in shared memory is periodically made consistent with data on disk; and that a buffer page that is written to disk is actually written in entirety.

Critical Sections

A *critical section* of code makes a set of disk modifications that must be performed as a single unit; either all the modifications must occur, or none can occur.

A thread that is in a critical section is holding shared-memory resources. Within the space of the critical section, the database server cannot determine which shared-memory resources should be released and which changes should be undone to return all data to a consistent point. Therefore, if a virtual processor is terminated while a thread is in a critical section, the database server takes the following two steps to ensure that all data is returned to the last known point of consistency:

- The database server aborts immediately.
- The database server initiates fast recovery the next time that it is initialized.

Fast recovery is the procedure that the database server uses to restore the physical and logical consistency of data quickly, up to and including the last record in the logical log. For a description of fast recovery, refer to [“Fast Recovery” on page 23-14](#).

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 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 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 checkpointed pages might not be consistent with each other because the database server does not flush all data pages to disk. 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
- 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.

Write-Ahead Logging and Fast Recovery

Fuzzy checkpoint depends on write-ahead logging for fast recovery to work correctly. *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 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.

For information on improving checkpoint performance, see the chapter about configuration impacts on I/O in your [Performance Guide](#). For information about the physical log, see [Chapter 21, “Physical Logging.”](#)

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 initiate a checkpoint from the command line with **onmode -c**
- When you initiate a checkpoint from the ON-Monitor Force-Ckpt menu ♦
- When you convert the database server to a newer version or revert to a previous version
- When you perform a backup or restore using **ontape**
- When you perform a backup or restore using ON-Bar
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.

Forcing a Full Checkpoint

You might want to force a full checkpoint for any of the following reasons, as well as others:

- You should initiate a full checkpoint 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).
- You should initiate a full checkpoint before you issue **onmode -sy** to place the database server in quiescent mode.
- You have just finished 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.
- You are about to attempt a system operation that might interrupt the database server. If a checkpoint has not occurred for a long time, fast recovery could take longer than you want.

- Foreground writes are taking more resources than you want. You can manually force a checkpoint to bring this down to zero for a while.
- You should initiate a full checkpoint before you execute **dbexport** or unload a table. The full checkpoint ensures the physical consistency of all data before you export or unload it.

To force a checkpoint, execute the following command from the command line:

```
onmode -c
```

Forcing a Fuzzy Checkpoint

To force a fuzzy checkpoint, execute the following command:

```
onmode -c fuzzy
```

Sequence of Events in a Checkpoint

The following section outlines the main events that occur during a checkpoint once a user thread raises the checkpoint-requested flag. 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 page-cleaner thread flushes the physical-log buffer.
4. In a fuzzy checkpoint, the page-cleaner threads flush modified pages for nonfuzzy operations in the buffer pool to disk.
In a full checkpoint, the page-cleaner threads flush all modified pages in the buffer pool to disk.
5. The checkpoint thread writes a checkpoint record to the logical-log buffer.
6. The physical log on disk 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 the checkpoint-requested flag is set, 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.

Page-Cleaner Thread Flushes the Physical-Log Buffer

After all threads have exited from critical sections, the page-cleaner thread resets the shared-memory pointer from the current physical-log buffer to the other buffer and flushes the buffer. After the buffer is flushed, the page-cleaner thread updates the time stamp that indicates the most recent point at which the physical-log buffer was flushed.

Page-Cleaner Threads Flush Modified Pages in the Buffer Pool

In a fuzzy checkpoint, the page-cleaner threads flush modified pages for nonfuzzy operations in the buffer pool to disk. They do not flush modified pages for fuzzy operations (inserts, deletes, updates) to disk. [Figure 23-1](#) shows how the database server writes only the nonfuzzy pages to disk. The shaded squares, marked **F**, represent the fuzzy pages.

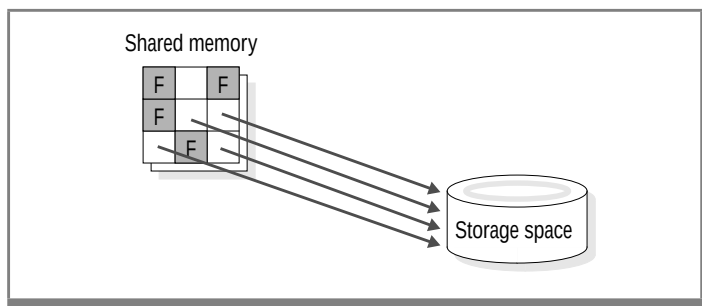


Figure 23-1
*Selectively Writing
Modified Pages
from Shared
Memory to Disk*

In a full checkpoint, the page cleaners flush all modified pages in the shared-memory buffer pool to disk. This flushing is performed as a chunk write.

Checkpoint Thread Writes Checkpoint Record

This step is the same for both fuzzy and full checkpoints. The page-cleaner thread writes a *checkpoint-complete* record to the logical-log buffer after the modified pages have been written to disk.

In a fuzzy checkpoint, the checkpoint thread also writes a dirty-pages table (DPT) record 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, see the *Backup and Restore Guide*.

For information on **ontape** or ON-Archive, see the [Archive and Backup Guide](#).

Time Stamps

The database server uses a time stamp to identify a time when an event occurred relative to other events of the same kind. The time stamp is not a literal time that refers to a specific hour, minute, or second. It is a 4-byte integer that the database server assigns sequentially. When the database server compares two time stamps, its algorithm accounts for the possibility that wraparound has occurred.

Time Stamps on Disk Pages

Each disk page has one time stamp in the page header and a second time stamp in the last 4 bytes on the page. The page-header and page-ending time stamps are synchronized after each write, so they should be identical when the page is read from disk. Each read compares the time stamps as a test for data consistency. If the test fails, an error is returned to the user thread, indicating either that the disk page was not fully written to disk or that the page has been partially overwritten on disk or in shared memory. For a description of the content of a dbspace page, refer to dbspace structure and storage in the chapter on disk structures and storage in the *Administrator's Reference*.

Time Stamps on Logical-Log Pages

Each logical-log record contains a time stamp, a 4-byte integer. During a restore, the database server uses these time stamps to determine where to start the logical restore.

For fuzzy operations, the database server uses the same time stamp for both the log record and disk page. During fast recovery, the database server compares the disk-page and log time stamps to determine which log record to apply. After fast recovery applies the log record to the dirty page, it copies the time stamp from the log record to the time stamp for the disk page.

Time Stamps on Blobpages

In addition to the page-header and page-ending time-stamp pair, each disk page that contains a simple large object (TEXT or BYTE data) also contains one member of a second pair of time stamps. This second pair of time stamps is the blobpage time-stamp pair. The time stamp that appears on the disk page where the simple large object is stored is paired with a time stamp that is stored with the forward pointer to this data segment. The forward pointer is stored either in the data row (with the descriptor) or with the previous segment of simple-large-object data. For more information on time stamps on blobpages, refer to the section on blobspace structure in the disk structure and storage chapter of the *Administrator's Reference*.

A blobpage time-stamp pair is updated whenever a TEXT or BYTE column is updated. When simple-large-object data in a row is updated, the new object is stored on disk, and the forward pointer that is stored with the descriptor is revised to point to the new location. The blobpage time stamp in the data row is updated and synchronized with the blobpage time stamp on the disk page of the new simple large object.

Blobpage Time Stamps with Dirty Read Isolation Level

Because simple large objects can be very large, it might be impossible to retrieve the data simultaneously with the rest of the row. Coordination is needed for simple-large-object reads at the Dirty Read or Committed Read level of isolation. Therefore, each read compares the two members of the blob time-stamp pair as a test for logical consistency of data. If the two time stamps in the pair differ, this inconsistency is reported as a part of consistency checking. The error indicates either that the pages have been corrupted or that the forward pointer is no longer valid.

To understand how a forward pointer for simple-large-object data might become invalid, consider the following examples.

A program using Dirty Read isolation can read rows that have been deleted, provided the deletion has not yet been committed. Assume that a user thread is deleting simple-large-object data from a row. During the delete process, another user thread that is operating with a Dirty Read isolation level reads the same row, searching for the blob-descriptor information. In the meantime, the first transaction completes, the simple-large-object data is deleted, the space is freed, and a third thread starts to write new simple-large-object data in the newly freed space where the first simple large object was stored. Eventually, when the second user thread starts to read the data at the location where the first object was stored, the thread compares the time stamp from the descriptor with the time stamp that precedes the data. The time stamps do not match. The time stamp on the blobpage is greater than the time stamp in the forward pointer, indicating to the user thread that the forward pointer information is obsolete.

Blobpage Time Stamps with Committed Read Isolation Level

If a program is using Committed Read isolation, the problem just described cannot occur because the database server does not see a row that has been marked for deletion. However, under Committed Read, no lock is placed on an undeleted row when it is read. Simple-large-object data is read in a second step, after the row is fetched. During this step, another program could delete the row, commit the deletion, and reuse the space on the disk page. If the space is reused in the interim, the time stamp for the simple-large-object data is greater than the time stamp in the forward pointer. In this case, the comparison indicates the obsolete pointer information, and the inconsistency is reported.

Fast Recovery

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time that it moves from off-line to quiescent mode or from off-line to on-line 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 off-line, 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 23-16.](#)

If the fast-recovery process finds that the database server came off-line in a controlled manner, the fast-recovery process terminates, and the database server moves to on-line 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. Such failures are usually caused by system failures. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Fast recovery addresses the following kinds of system failure:

- The database server is processing tasks for more than 40 users.
- Dozens of transactions are in process.
- Without warning, the operating system fails.

How does the database server bring itself to a consistent state again? What happens to ongoing transactions? The answer to both questions is fast recovery.

Situations When Fast Recovery Is Initiated

Every time that the administrator brings the database server to quiescent mode or on-line mode from off-line 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 on-line 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 off-line 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 17-13](#)), 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.)

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. Use the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint.
2. Locate the most recent checkpoint record in the logical-log files.
3. Roll forward all logical-log records written after the most recent checkpoint record.
4. Roll back transactions that do not have an associated COMMIT or BEGIN COMMIT record in the logical log.

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 23-2](#) illustrates this step.

The database server is now physically consistent.

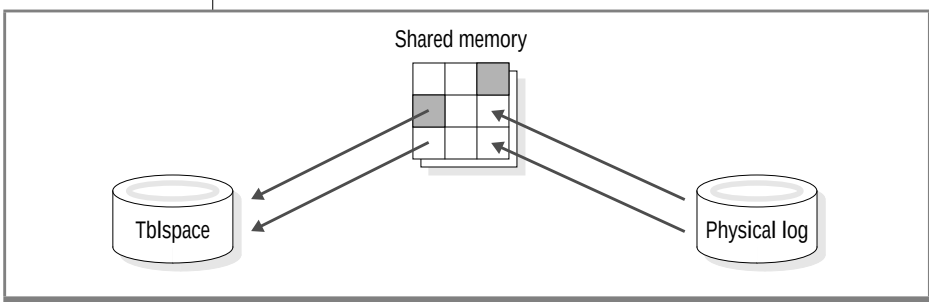


Figure 23-2
*Writing All
Remaining Before-
Images in the
Physical Log Back
to Disk*

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. [Figure 23-3](#) illustrates this step.

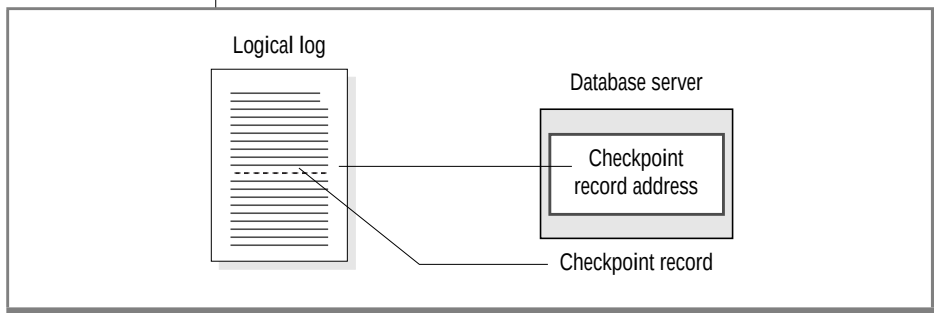


Figure 23-3
Locating the Most Recent Checkpoint Record in the Logical Log

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 23-4](#) illustrates this step.

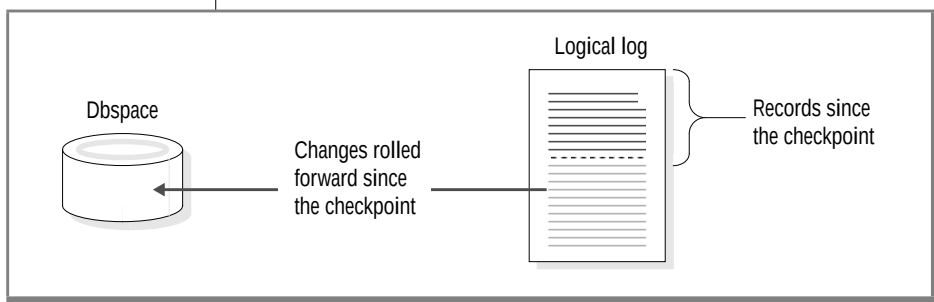


Figure 23-4
Rolling Forward the Logical-Log Records Written Since the Most Recent Checkpoint

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 30-10.

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 23-5](#) illustrates the rollback procedure. When fast recovery is complete, the database server goes to quiescent or on-line mode.

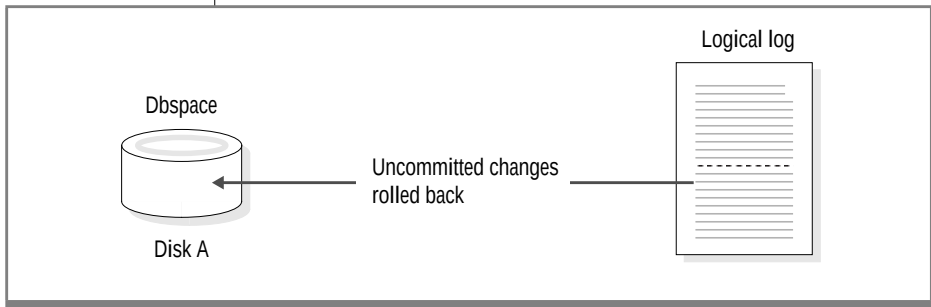


Figure 23-5
*Rolling Back All
Incomplete
Transactions*

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.

These stages can also be expressed as the following steps, which are described in detail in the paragraphs that follow:

1. Use 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. Locate the oldest update in the logical-log that is not yet flushed to disk.
3. Apply the log records for fuzzy operations that occurred before the most recent checkpoint.
4. Roll forward all logical-log records written after the most recent checkpoint record.
5. Roll 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, returning 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 23-2 illustrates this step.

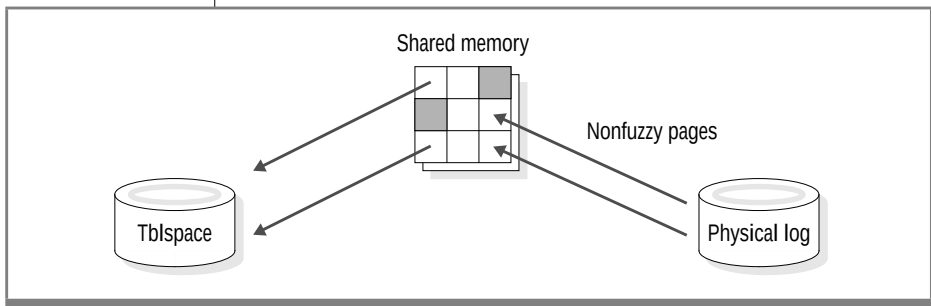


Figure 23-6
*Writing Nonfuzzy
Before-Images in
the Physical Log
Back to Disk*

Pages on which fuzzy operations occurred are not physically consistent because the database server does not physically log their 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

In this step of fast recovery, 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. The database server no longer starts fast recovery at the most recent checkpoint record.

Figure 23-7 shows that the oldest update in the logical log occurred several checkpoints ago and that all the log records are applied.

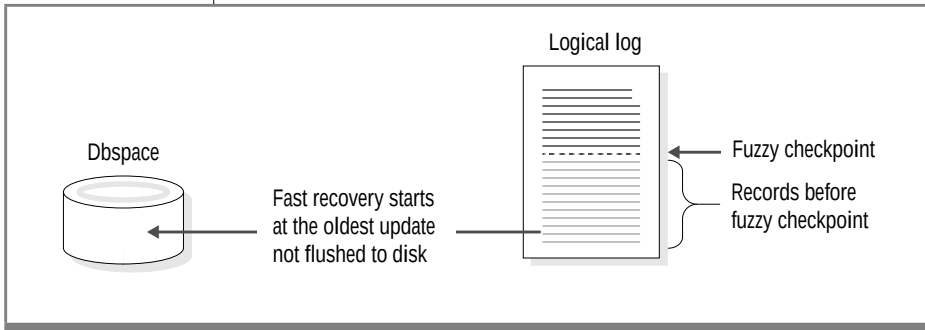


Figure 23-7
Locating the Oldest Update Record in the Logical Log

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 23-8 illustrates how the database server processes fuzzy records only prior to checkpoint.

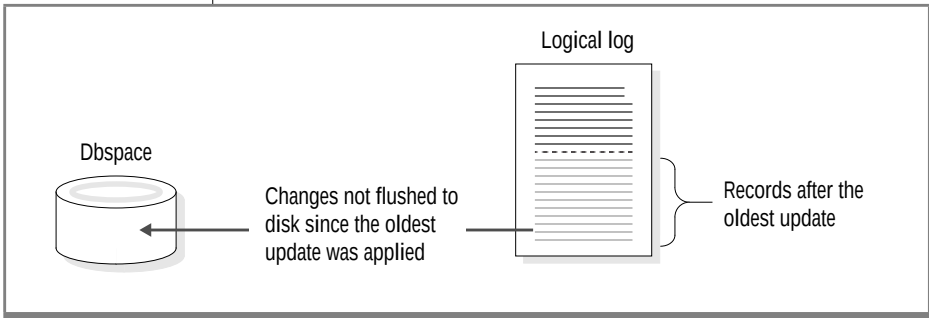


Figure 23-8
*Applying the Log
Records for Fuzzy
Operations*

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 23-9 illustrates the roll forward of all records after the fuzzy checkpoint.

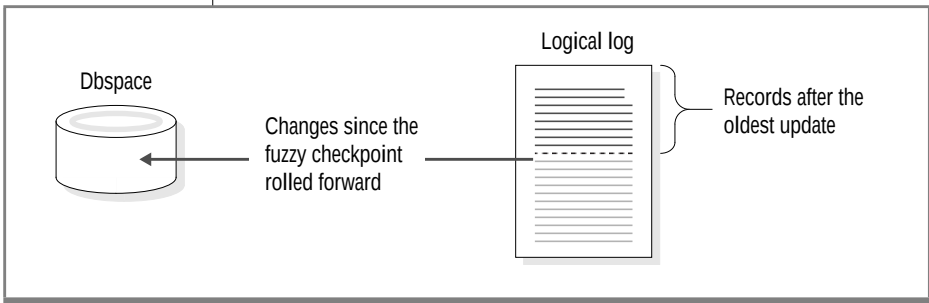


Figure 23-9
*Rolling Forward the
Logical-Log
Records Written
Since the Most
Recent Fuzzy
Checkpoint*

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 23-10](#) illustrates the rollback procedure. When fast recovery is complete, the database server goes to quiescent or on-line mode.

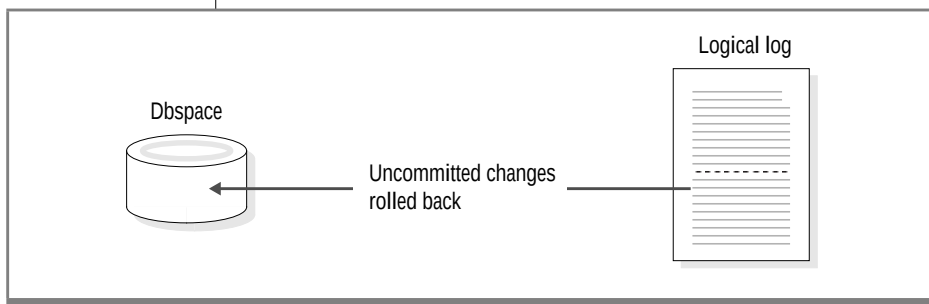


Figure 23-10
*Rolling Back All
Incomplete
Transactions*

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](#).

Using onstat Options

You can use the following **onstat** options to obtain checkpoint information:

- **-m**
- **-p**

Execute **onstat -m** to view the last 20 entries in the message log. If a checkpoint record does not appear in the last 20 entries, read the message log directly with a text editor. The database server writes individual checkpoint records to the log when the checkpoint ends. If a checkpoint check occurs, but the database server has no pages to write to disk, the database server does not write any records to the message log.

UNIX

Execute **onstat -p** to obtain these checkpoint statistics:

- Number of checkpoints that occurred since the database server was brought on-line (**numckpts**)
- Number of times that a user thread waits for a checkpoint to finish (**ckpwaits**)

The database server prevents a user thread from entering a *critical section* during a checkpoint.

Using ON-Monitor

You can use the following ON-Monitor options to monitor checkpoint information.

Profile Option

Select the Profile option from the Status menu. This option displays the **Checkpoints** and **Check Waits** fields described in the previous section about **onstat -p**. The **Checkpoints** and **Check Waits** fields in ON-Monitor display the same information as the **numckpts** and **ckpwaits** fields in **onstat -p**.

Force-Ckpt Option

Select the Force-Ckpt menu. [Figure 23-11](#) shows the screen display.

```
Do you want to force a checkpoint? (y/n)
```

```
Last checkpoint done   :   Fri Jul 30 09:34:33 1999
Last checkpoint check  :   Fri Jul 30 13:16:50 1999
```

Figure 23-11
ON-Monitor
Force-Ckpt Screen

A checkpoint check occurs if the time specified by the CKPTINTVL configuration parameter has elapsed since the last checkpoint. If no modifications have been made since the time of the last checkpoint, the database server does not perform a checkpoint; that is, the database server does not flush the physical-log buffers to disk. The time in the **Last Checkpoint Done** field does not change until a checkpoint occurs.

Using SMI Tables

The **sysprofile** table provides the same checkpoint statistics that are available from the following sources:

- **onstat -p** option
- Profile option in ON-Monitor ♦

The **sysprofile** table contains two columns, **name** and **value**. The **name** column contains the statistic name, and the **value** column contains the statistic value. These rows contain the following checkpoint information.

Checkpoint Statistic	Description
numckpts	Number of checkpoints that have occurred since the database server was brought on-line
ckptwaits	Number of times that threads waited for a checkpoint to finish to enter a <i>critical section</i> during a checkpoint

Fault Tolerance

- Chapter 24** **Mirroring**
- Chapter 25** **Using Mirroring**
- Chapter 26** **High Availability on Windows NT**
- Chapter 27** **High-Availability Data Replication**
- Chapter 28** **Using High-Availability Data Replication**
- Chapter 29** **Consistency Checking**



Mirroring

Mirroring	24-4
Benefits of Mirroring	24-4
Costs of Mirroring.	24-5
Consequences of Not Mirroring	24-5
Data to Mirror	24-5
Alternatives to Mirroring	24-6
Logical Volume Managers.	24-6
Hardware Mirroring.	24-7
External Backup and Restore.	24-7
Mirroring Process	24-7
Creation of a Mirrored Chunk.	24-7
Mirror Status Flags	24-8
Recovery	24-9
Actions During Processing	24-9
Disk Writes to Mirrored Chunks	24-9
Disk Reads from Mirrored Chunks.	24-10
Detection of Media Failures	24-10
Chunk Recovery	24-11
Result of Stopping Mirroring	24-11
Structure of a Mirrored Chunk	24-11

In This Chapter

The first part of this chapter answers the following basic questions about the database server mirroring feature:

- What are the benefits of mirroring?
- What are the costs of mirroring?
- What happens if you do not mirror?
- What should you mirror?
- What mirroring alternatives exist?

The second part of the chapter discusses the actual mirroring process. The following aspects of the process are discussed:

- What happens when you create a mirrored chunk?
- What are the mirror status flags?
- What is recovery?
- What happens during processing?
- What happens if you stop mirroring?
- What is the structure of a mirrored chunk?

For instructions on how to perform mirroring tasks, refer to [Chapter 25, “Using Mirroring.”](#)

Mirroring

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace, blobspace, or sbospace 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 24-1](#). 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.

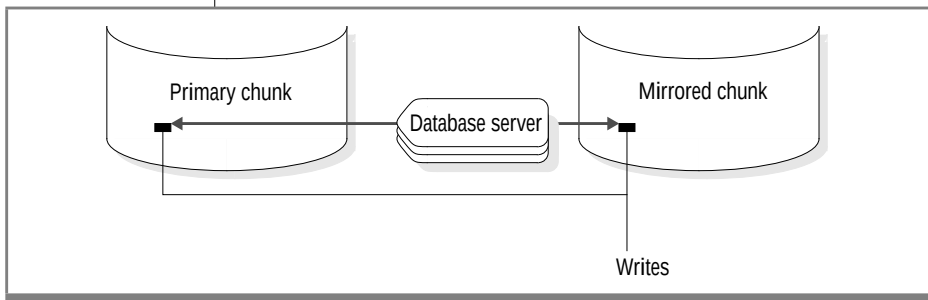


Figure 24-1
*Writing Data to Both
the Primary Chunk
and the Mirrored
Chunk*

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 a media failure occurs, mirroring provides the database server administrator with a means of recovering data without having to take the database server off-line. 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 24-9](#).

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 a media failure increases.

When a mirrored chunk suffers a media failure, the database server reads exclusively from the chunk that is still on-line until you bring the down chunk back on-line. 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 off-line 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 off-line 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 other chunks that store frequently used data. This action ensures that the activities of many users are not 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 24-9.](#)) In addition, database server mirroring uses split reads to improve read performance. (See [“Disk Reads from Mirrored Chunks” on page 24-10.](#)) 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 in a manner resilient to media failure. The disadvantage is that it is slower than database server mirroring for write operations.

External Backup and Restore

If you use hardware mirroring from a manufacturer such as EMC, consider using external backups and restores. In an external backup, your site can copy the data from mirrored disks to another location, and then swap these disks back to their original locations when you need to restore the data. For more information on external backup and restore, see the *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 on, refer to [Chapter 25, “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 25, “Using Mirroring.”](#)

Mirror Status Flags

Dbspaces, blobspaces, and sbspaces have status flags that indicate whether it is mirrored, unmirrored, or mirrored.

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 on-line, down, a new mirrored chunk that requires a level-0 backup of the root dbspace, or 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 15-46](#).

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 on-line chunk onto the new, repaired chunk until the two are considered identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the on-line chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives on-line 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 on-line chunk while the recovery process is occurring. 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 25-12](#).

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 writes for each modification: one to the primary chunk and one to the mirrored chunk. Virtual processors of the AIO class perform the actual disk I/O. For more information, refer to “[Asynchronous I/O](#)” on [page 10-29](#).

The requesting thread submits the two write requests (one for the primary chunk and one for the mirrored chunk) asynchronously. That is, if two AIO virtual processors are idle, they can perform the two disk writes in parallel. In the meantime, the requesting thread can perform any additional processing that does not depend on the result of the mirror I/O.

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 24-2 on page 24-10](#) illustrates a split read.

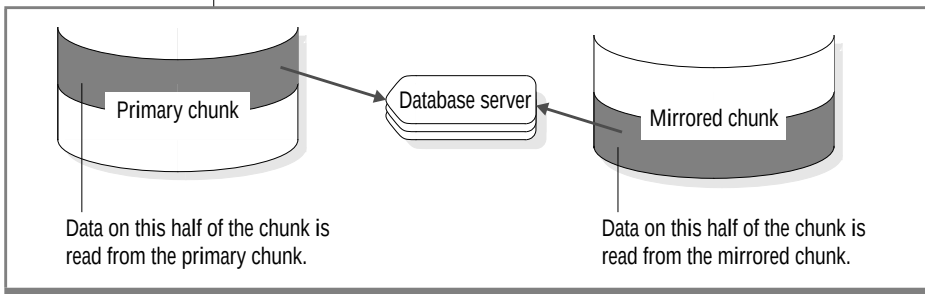


Figure 24-2
*Split Read Reducing
the Maximum
Distance Over
Which the Disk Head
Must Travel*

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 24-8](#).

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains on-line. 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 on-line 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 on-line 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 15-46](#), 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 on-line.

Structure of a Mirrored Chunk

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

Steps Required for Mirroring Data	25-3
Enabling Mirroring	25-4
Changing the MIRROR Parameter with ONCONFIG.	25-4
Changing the MIRROR Parameter with ON-Monitor.	25-5
Allocating Disk Space for Mirrored Data	25-5
Linking Chunks	25-6
Relinking a Chunk to a Device After a Disk Failure	25-6
Using Mirroring	25-7
Mirroring the Root Dbspace During Initialization	25-8
Changing the Mirror Status	25-8
Taking Down a Mirrored Chunk	25-8
Ending Mirroring	25-9
Managing Mirroring	25-9
Starting Mirroring for Unmirrored Dbspaces	25-9
Starting Mirroring for Unmirrored Dbspaces with onspaces	25-9
Starting Mirroring for Unmirrored Dbspaces with ON-Monitor.	25-10
Starting Mirroring for New Dbspaces	25-10
Starting Mirroring for New Dbspaces with onspaces	25-10
Starting Mirroring for New Dbspaces with ON-Monitor	25-10
Adding Mirrored Chunks	25-11
Adding Mirrored Chunks with the onspaces Utility	25-11
Adding Mirrored Chunks with ON-Monitor	25-11
Taking Down a Mirrored Chunk	25-11
Taking Down a Mirrored Chunk with the onspaces Utility	25-11
Taking Down a Mirrored Chunk with ON-Monitor	25-12

Recovering a Mirrored Chunk	25-12
Recovering a Mirrored Chunk with onspaces	25-12
Recovering a Mirrored Chunk with ON-Monitor	25-12
Ending Mirroring	25-13
Ending Mirroring with onspaces	25-13
Ending Mirroring with ON-Monitor	25-13

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. Then it describes the following tasks:

- Enabling mirroring
- Allocating disk space for mirrored chunks
- Starting mirroring (creating mirrored chunks)
- Adding chunks to mirrored dbspaces
- Changing the mirror status of chunks
- Relinking mirrored chunks after a disk failure
- Ending mirroring

Steps Required for Mirroring Data

To start mirroring data on a database server that is not running with the mirroring function enabled, you must perform the following steps:

1. Take the database server off-line and enable mirroring. See [“Enabling Mirroring” on page 25-4](#).
2. Bring the database server back on-line.
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. See [“Allocating Disk Space for Mirrored Data” on page 25-5](#).

4. Choose the dbspace, blob space, or sb space that you want to mirror, and create mirrored chunks by specifying 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 25-7](#).

Enabling Mirroring

When you enable mirroring, you invoke 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 sb space. See [“Using Mirroring” on page 25-7](#).

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.

Changing the MIRROR Parameter with ONCONFIG

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 while the database server is in on-line mode. After you change the ONCONFIG file, reinitialize shared memory (take the database server off-line and then to quiescent mode) for the change to take effect.

UNIX

Changing the MIRROR Parameter with ON-Monitor

If you are logged in on a UNIX system as user **informix** or **root**, you can use ON-Monitor to change the value of MIRROR. Informix recommends that you edit the ONCONFIG file to enable mirroring. If you are not careful when you enable mirroring with ON-Monitor, you might accidentally reinitialize your disk.

To enable mirroring, choose Parameters, Initialize. In the field that is labelled **Mirror**, enter Y. Press ESC to record changes.

A series of screens appears displaying other system parameters. Type ESC at each screen to maintain the same values. 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.*

Reinitialize shared memory (take the database server off-line and then to quiescent mode) for the change to take effect.

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 15-6](#).

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.

UNIX

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 on-line.

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 on-line. 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 -lg  
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.

This action consists of specifying disk space that you have already allocated, either raw disk space or a cooked file, for each mirrored chunk.

When you create a mirrored chunk, the database server performs the *recovery* process, copying 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 perform the recovery process 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 24-7](#).

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.

When you select a space to mirror, you must create mirrored chunks for every chunk within 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 on-line 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 on-line.

For information on how to determine the status of a chunk, refer to [“Monitoring Disk Usage” on page 15-46](#).

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 25-6](#).)

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.

Ending Mirroring

When you end mirroring for a dbspace or blobspace, the database server immediately releases the mirrored chunks of that dbspace or blobspace. These chunks are immediately available for reassignment to other dbspaces or blobspaces.

Only users **informix** and **root** on a UNIX system or members of the **Informix-Admin** group on a Windows NT system can initiate this action.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in on-line mode when you end mirroring.

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 Dbspaces

You can prepare mirroring for a dbspace, , or sbpace at any time. However, the mirroring does not start until you perform a level-0 backup.

Starting Mirroring for Unmirrored Dbspaces with onspaces

You can use the **onspaces** utility to start mirroring a dbspace, blobspace, or sbpace. 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
```

UNIX

Starting Mirroring for Unmirrored Dbspaces with ON-Monitor

Use the Mirror option of the Dbspaces menu to start mirroring a dbspace. The first screen displays a list of dbspaces. To select the dbspace that you want to mirror, move the cursor down the list to the correct dbspace and type 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 unless they contain logical-log files, in which case recovery is postponed until after you create a level-0 backup.

Starting Mirroring for New Dbspaces

You can also start mirroring when you create a new dbspace, blobspace, or sbospace.

Starting Mirroring for New Dbspaces 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
```

UNIX

Starting Mirroring for New Dbspaces with ON-Monitor

To create a dbspace with mirroring, choose the Create option of the Dbspaces menu. 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.

Adding Mirrored Chunks with the onspaces Utility

You can also 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
```

UNIX

Adding Mirrored Chunks with ON-Monitor

In ON-Monitor, the Add-chunk option of the Dbspaces menu 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

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 25-6.)

Taking Down a Mirrored Chunk with the onspaces Utility

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

Taking Down a Mirrored Chunk with ON-Monitor

To use ON-Monitor to take down a mirrored chunk, choose the Status option from the Dbspaces menu. 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

You recover a down chunk to begin mirroring the data in the chunk that is on-line.

Recovering a Mirrored Chunk with onspaces

You can use the **onspaces** 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 -0
```

UNIX

Recovering a Mirrored Chunk with ON-Monitor

To use ON-Monitor to recover a down chunk, choose the Status option from the Dbspaces menu. With the cursor on the dbspace that contains the down chunk, press F3 or CTRL-B. The system displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that is down, and type F3 or CTRL-B to recover it.

Ending Mirroring

When you end mirroring for a dbspace, blobspace, or sbospace, the database server immediately releases the mirrored chunks of that space.

Ending 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
```

Ending Mirroring with ON-Monitor

To end mirroring for a dbspace or blobspace with ON-Monitor, select the Mirror option of the Dbspaces menu. Select a dbspace or blobspace that is mirrored, and type CTRL-B or F3.

High Availability on Windows NT

Overview of the MSCS Feature.	26-3
How the Database Server Uses MSCS	26-4
Multiple Residency	26-5
Client Connections	26-5
Automatic Backups	26-6
Database Server Administration with MSCS.	26-6
The sqlhosts Registry.	26-6

In This Chapter

The database server supports the Microsoft Cluster Server (MSCS) feature, which enables high availability on Windows NT. High availability provides protection against single points of failure for database server resources.

Overview of the MSCS Feature

The MSCS feature allows you to cluster two Windows NT computers, referred to in the MSCS feature as *nodes*, as redundant components. When a failure occurs on one node in the cluster, Windows NT restarts the failed applications on the surviving node in the pair. The Microsoft Cluster Manager and Resource Manager provide the following capabilities:

- Failure detection
- Communication of failure to other subsystems or applications
- System level recovery
- Restarting the cluster-aware application on the surviving node

To obtain failure protection, an application registers a dynamically linked library (DLL) for the high-availability resource with the Windows NT Resource Manager. The Windows NT Resource Manager uses the DLL to invoke specific start, restart, stop, and monitoring functions for the resource.

A *resource* is a hardware component, such as a shared disk, or a software application that is shared between the two nodes in a cluster. When a failure occurs on one node, the Resource Manager moves the resource over to the surviving node and restarts it.

The database server implements high availability with MSCS by sharing one or more hard disks in a cluster. At any given time, only one node owns the shared disks. The other node accesses the shared disks only in the event of failure on the first node.

MSCS includes an administration tool called the Cluster Administrator. The Cluster Administrator enables you to designate a cluster and define resources, resource ownership, and dependencies on other resources. The Cluster Administrator also enables you to define groups that specify resource dependencies, allowing the Resource Manager to move groups of dependent resources to the surviving node in the event of failure.

To a client application, the cluster appears to be a single host computer that has one name, the cluster name. However, each node also has its own name and can have its own private resources.

How the Database Server Uses MSCS

When you install the database server, you place all dbspaces on a disk that both nodes of a cluster share. Although the database server must be installed on both nodes, it runs on only one node at a time.

If one of the following components of the computer that is running the database server fails, MSCS can automatically restart the database server on the other node:

- The computer
- The network connection
- The disk controller card

When a failure occurs, the Resource Manager restarts the database server on the surviving node, without operator intervention. Because the dbspaces are on shared disks, no restore or application of transaction logs is necessary.

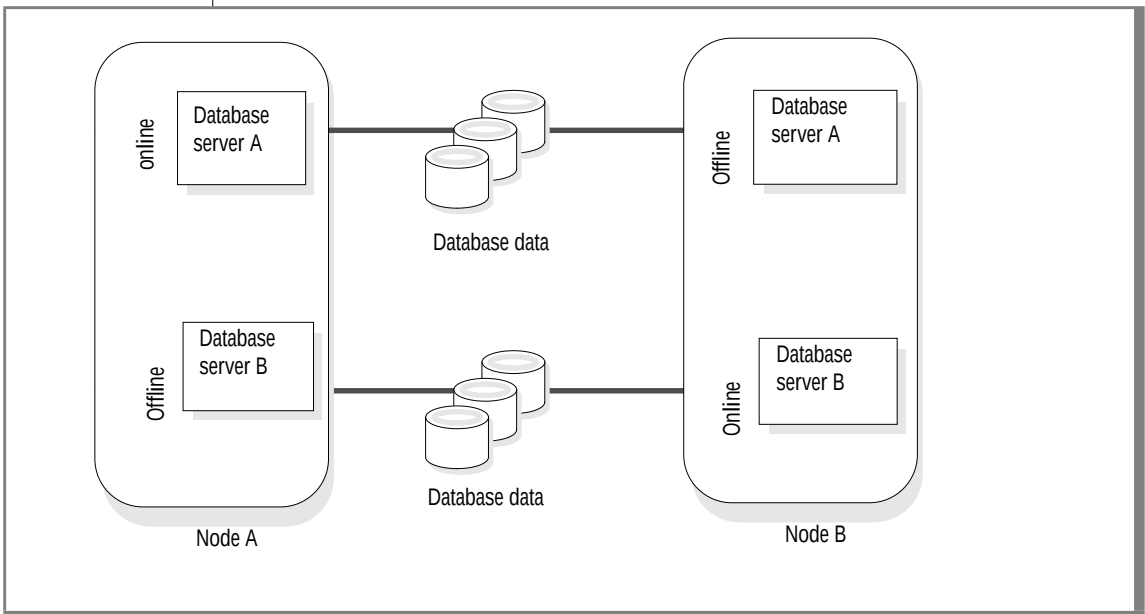
If the database server fails but the computer remains running, MSCS restarts the database server on the same node.

Multiple Residency

Because an instance of the database server that uses MSCS runs on only one node at a time, you can use the other node for other purposes at the same time. You can even run another database server on the second node.

Figure 26-1 shows two instances of the database server, each running on a separate node under normal circumstances. If node A becomes unavailable, database server A automatically starts running on node B. If node B becomes unavailable, database server B automatically starts running on node A.

Figure 26-1
Database Servers Using MSCS



Client Connections

The **sqlhosts** registry contains the connection information that the application uses to connect. Instead of listing a machine name for the connection, the **sqlhosts** registry lists only a cluster name. MSCS determines which node is currently running the database server when the client application attempts a connection.

When a node that is running the database server fails, MSCS automatically starts the database server on the surviving node. Because the database server has been restarted, any transaction in progress when the switch occurs is rolled back. The application must reconnect to the database server.

Automatic Backups

If automatic log backups are enabled, set ISMDIR and PATH in the system environment as follows:

```
ISMDIR=<drive_letter>:\ISM\1.00  
PATH=...;<drive_letter>:\ISM\1.00\bin
```

Append the **ISM\1.00\bin** directory to the existing path.

Database Server Administration with MSCS

No additional administrative tasks are required to use the database server with MSCS. However, some general administrative tasks are different when your database server uses MSCS:

- Use the Microsoft Cluster Administrator to start the database server. Although you can start the database server with **oninit** or with the Services tool, the preferred method is the Microsoft Cluster Administrator.
- Complete an additional step when you create a new database server instance.

When you create a new instance of the database server, add that instance as a resource, part of the Informix group.

The sqlhosts Registry

Informix recommends that you establish a global **sqlhosts** registry when you install the database server. When you create a global **sqlhosts** registry, connection information is shared between computers. If you do not use a global **sqlhosts** registry, you must make sure connection information is the same for both nodes.

High-Availability Data Replication

Data Replication	27-4
High-Availability Data Replication	27-4
Primary and Secondary Database Servers	27-5
High-Availability Data Replication Versus Mirroring	27-6
High-Availability Data Replication Versus Two-Phase Commit	27-8
How High-Availability Data Replication Works	27-8
How Data Initially Replicates	27-8
Reproduction of Updates to the Primary Database Server	27-9
How the Log Records Are Sent	27-10
High-Availability Data-Replication Buffers	27-10
When Log Records Are Sent	27-11
Synchronous Updating	27-11
Asynchronous Updating	27-11
Threads That Handle High-Availability Data Replication	27-14
Checkpoints Between Database Servers	27-15
How Data Synchronization Is Tracked	27-15
High-Availability Data-Replication Failures	27-16
High-Availability Data-Replication Failures Defined	27-16
Detection of High-Availability Data-Replication Failures	27-17
Actions When a High-Availability Data-Replication Failure	
Is Detected	27-18
Considerations After High-Availability Data-Replication Failure.	27-18
Actions to Take If the Secondary Database Server Fails.	27-19
Actions to Take If the Primary Database Server Fails	27-19

Redirection and Connectivity for Data-Replication Clients	27-20
Designing Clients for Redirection	27-21
Automatic Redirection with DBPATH	27-21
How the DBPATH Redirection Method Works	27-21
What the Administrator Needs to Do	27-22
What the User Needs to Do	27-22
Administrator-Controlled Redirection with the Connectivity	
Information	27-23
How Redirection with the Connectivity Information Works	27-23
What the Administrator Needs to Do	27-24
What the User Needs to Do	27-27
User-Controlled Redirection with INFORMIXSERVER	27-27
How Redirection Works with INFORMIXSERVER	27-27
What the Administrator Needs to Do	27-28
What the User Needs to Do	27-28
Handling Redirection Within an Application.	27-28
Comparison of Different Redirection Mechanisms	27-30
Designing High-Availability Data-Replication Clients	27-31
Setting Lock Mode to Wait for Access to Primary Database Server	27-31
Designing Clients to Use the Secondary Database Server	27-32
No Data Modification Statements	27-32
Locking and Isolation Level	27-34
Use of Temporary Dbspaces for Sorting and Temporary Tables	27-35

In This Chapter

This chapter describes high-availability data replication for Dynamic Server.

High-availability data replication provides synchronous and asynchronous data replication of an entire database server. Use high-availability data replication if you require a hot standby. If you want to use asynchronous data replication, see the [Guide to Informix Enterprise Replication](#).

The chapter discusses the following topics:

- What data replication is, both in a broad sense and in the context of the database server
- How high-availability data replication works
- How high-availability data replication handles failures
- How the system administrator or user can redirect a client to connect to the other database server in the high-availability data-replication pair
- What the design considerations are for applications that connect to the secondary database server

A companion chapter, [Chapter 28, “Using High-Availability Data Replication,”](#) contains instructions on how to accomplish the administrative tasks that are involved in using high-availability data replication.

Data Replication

Data replication, in the broadest sense, refers to the process of representing database objects at more than one distinct site.

For example, 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.

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.

High-Availability Data Replication

Within the broad framework of data replication, 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. High-availability data replication 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.

Primary and Secondary Database Servers

When you configure a pair of database servers to use high-availability data replication, 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 high-availability data replication 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 27-1](#) illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.

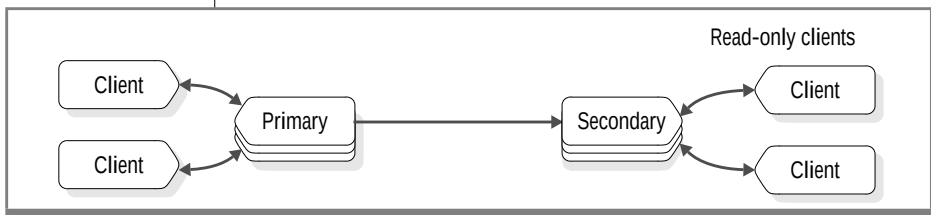


Figure 27-1
A Primary and Secondary Database Server in a High-Availability Data-Replication Pair

If one of the database servers fails, as [Figure 27-2](#) shows, you can redirect the clients that use that database server to the other database server in the pair.

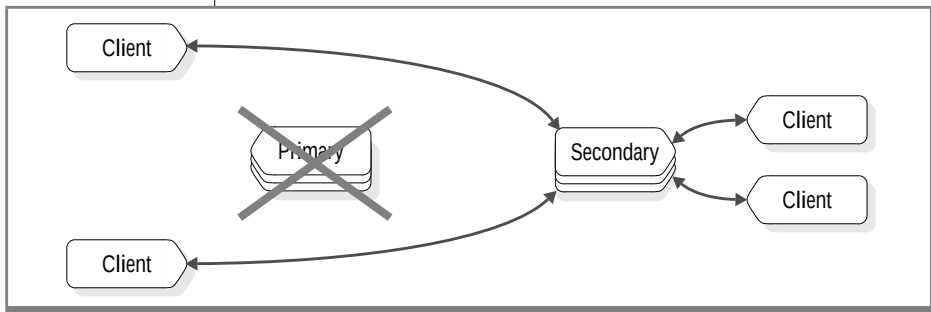


Figure 27-2
Database Servers in a High-Availability Data-Replication Pair and Clients After a Failure

If a primary database server fails, you can change the secondary database server to a standard database server so that it can accept updates.

High-availability data replication has the following features:

- Provides for quick recovery if one database server experiences a failure
- Allows for load balancing across the two database servers

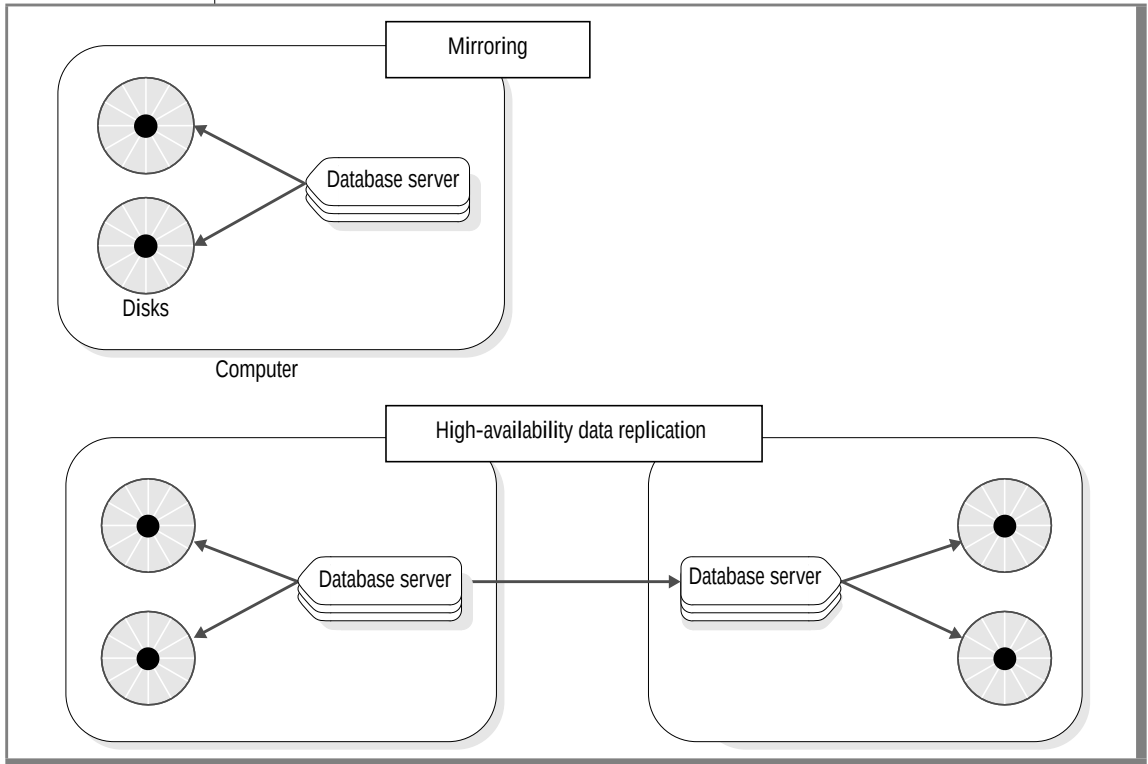
High-Availability Data Replication Versus Mirroring

High-availability data replication and mirroring are both transparent ways of making the database server more fault tolerant. However, as shown in [Figure 27-3 on page 27-7](#), they are quite different.

Mirroring, described in [“Mirroring” on page 24-4](#), 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, high-availability data replication duplicates on an entirely separate database server all the data that a database server manages (not just specified dbspaces). Because high-availability data replication 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 27-3
A Comparison of Mirroring and High-Availability Data Replication



High-Availability Data Replication Versus Two-Phase Commit

The two-phase commit protocol, described in detail in [Chapter 30](#), “[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 dynamic high-availability data replication.

How High-Availability Data Replication Works

This section describes the mechanisms that the database server uses to perform high-availability data replication. For instructions on how to set up, start, and administer a high-availability data-replication system, refer to [Chapter 28](#), “[Using High-Availability Data Replication](#).”

How Data Initially Replicates

The database server uses dbspace backups and logical-log files (both those backed up to tape and those on disk) to perform an initial replication of the data on one database server to a second database server. The procedure is basically as follows:

1. To make the bulk of the data managed by the two database servers the same, create a level-0 backup of all the dbspaces on one database server, and restore all the dbspaces from that backup on the other database server in the data-replication pair.
2. The database server that you restored from a dbspace backup in the first step then reads all the logical-log records generated since that dbspace backup from the database server on which the backup was created. 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 High-Availability Data Replication for the First Time” on page 28-11](#). The *Backup and Restore Guide* discusses how to initialize replication using ON-Bar. The *Archive and Backup Guide* discusses how to initialize replication using ON-Archive.

You must perform the initial high-availability data replication with a dbspace 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 high-availability data replication to work.

In the preceding steps, the database server from which you create the dbspace backup can be the primary database server or the secondary database server.

When high-availability data replication is working, the primary database server is in on-line mode 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

High-availability data replication 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. (For general information on transaction logging, refer to [“Transaction Logging” on page 17-6](#).) 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. The database server does not replicate data in blobspaces either.*

How the Log Records Are Sent

As shown in [Figure 27-4 on page 27-10](#), 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.

High-Availability Data-Replication Buffers

The high-availability data-replication buffers are part of the virtual shared memory that the primary database server manages. The high-availability data-replication buffers hold logical-log records before the primary database server sends them to the secondary database server. The high-availability data-replication buffers are the same size as the logical-log buffers.

[Figure 27-4](#) shows this concept.

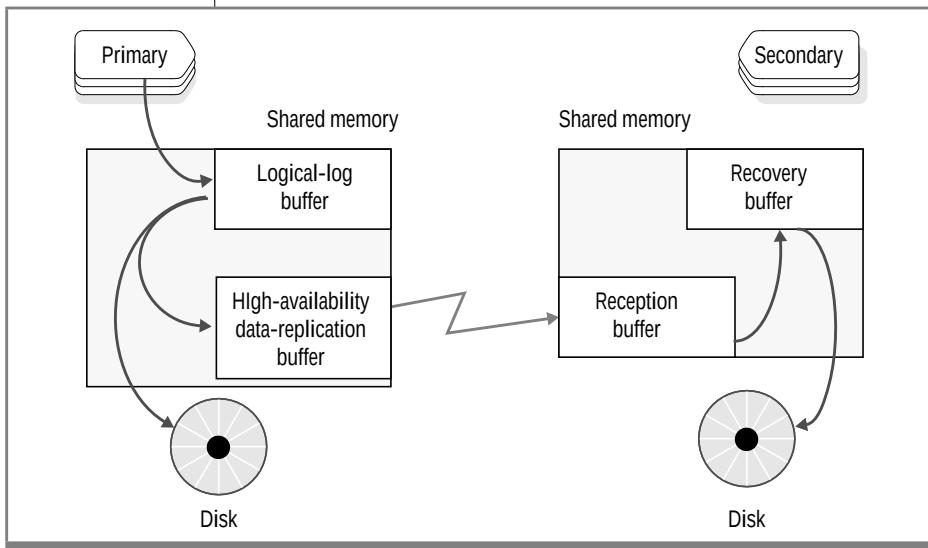


Figure 27-4
*How Logical-Log
Records Are Sent
from the Primary
Database Server to
the Secondary
Database Server*

When Log Records Are Sent

The primary database server sends the contents of the high-availability data-replication 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, high-availability data replication occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the high-availability data-replication buffer, it sends those records from the high-availability data-replication 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 high-availability data-replication buffer. Independent of that action, the primary database server sends the contents of the high-availability data-replication buffer across the network when one of the following conditions occurs:

- The high-availability data-replication 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.

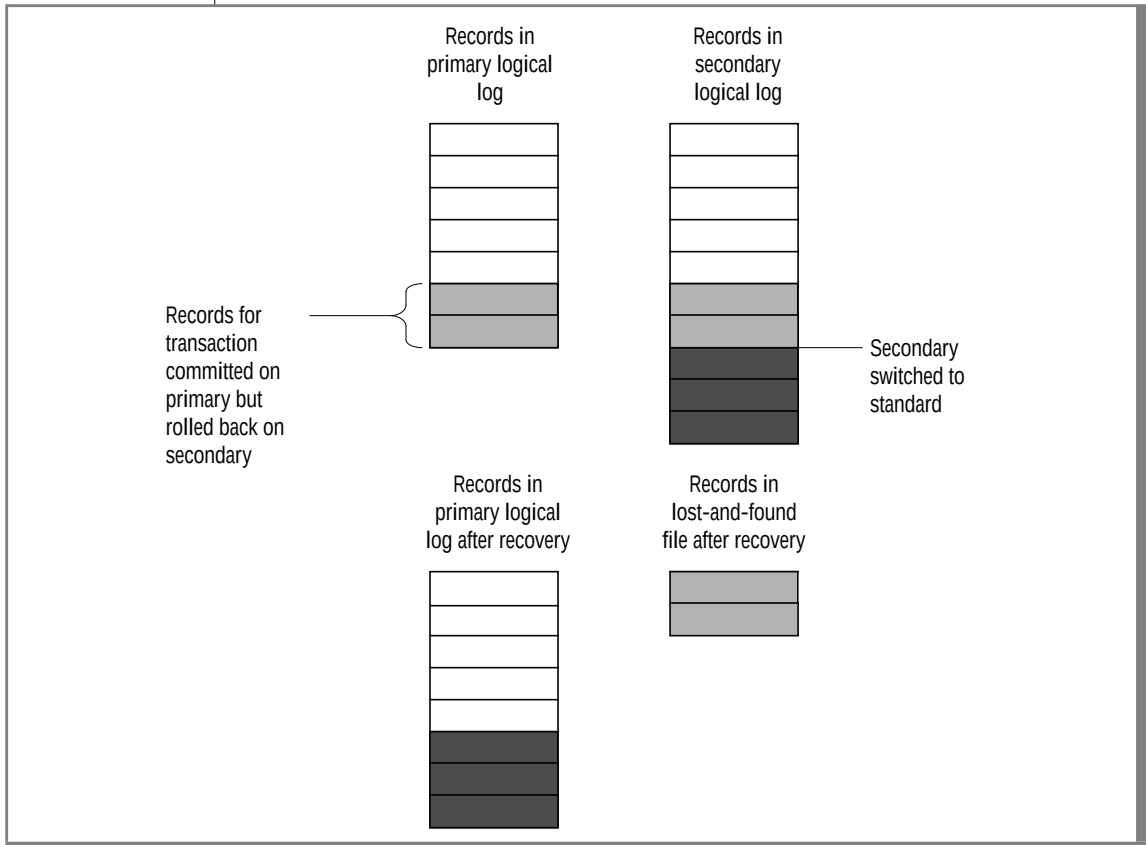
This method of updating might provide better performance than synchronous updating. However, as the following section explains, transactions might be lost.

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 high-availability data-replication 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 27-5](#) illustrates the process.

Figure 27-5
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 primary database server that is writing the transaction records to disk and the primary database server that is sending these records to the secondary database server.

Threads That Handle High-Availability Data Replication

The database server starts specialized threads to support data replication. As [Figure 27-6](#) shows, a thread called **drprsend** on the primary database server sends the contents of the high-availability data-replication buffer across the network to a thread called **drsecrcv** on the secondary database server.

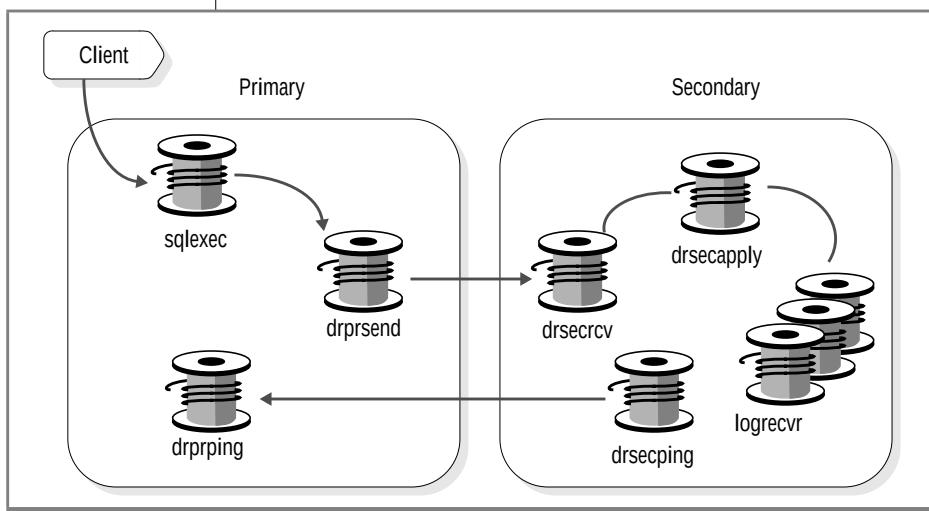


Figure 27-6
*Threads That
Manage Data
Replication*

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 `ON_RECVRY_THREADS` specifies the number of **logrecvr** threads used.

The remaining threads that the database server starts for high-availability data replication are the **drprping** and **drsecping** threads, which are responsible for sending and receiving the signals that indicate if the two database servers are connected.

Checkpoints Between Database Servers

Checkpoints between database servers in a high-availability data-replication pair are synchronous, regardless of the value of DRINTERVAL. (See [“Checkpoints” on page 23-4](#).) 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 [“High-Availability Data-Replication Failures Defined” on page 27-16](#).

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 reserve 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.

High-Availability Data-Replication Failures

This section discusses the causes and consequences of a high-availability data-replication failure, as well as the administrator's options for managing failure and restarting data replication.

High-Availability Data-Replication Failures Defined

A high-availability data-replication failure is a loss of connection between the database servers in a high-availability data-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
- An administrative action to turn off data replication on one of the database servers (that is, changing the type of the database server to standard)
- A disk failure on the secondary database server that is not resolved by a mirrored chunk



Tip: A high-availability data-replication failure does not necessarily mean that one of the database servers has failed, only that the high-availability data-replication connection between the two database servers is lost.

Detection of High-Availability Data-Replication Failures

The database server interprets either of the following conditions as a high-availability data-replication failure:

- A specified time-out value was exceeded.

In the course of normal high-availability data-replication 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 a high-availability data-replication failure has occurred.

- The periodic signaling (*pinging*) of the other database server over the network does not yield response.

Both database servers send a signal (or *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 signal 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 signal attempts in a row, the database server that was signaling assumes that a high-availability data-replication failure has occurred.

Actions When a High-Availability Data-Replication Failure Is Detected

After a database server detects a high-availability data-replication failure, it writes a message to its message log (for example, DR: receive error) and turns data replication off. If a high-availability data-replication failure occurs, the high-availability data-replication connection between the two database servers is dropped and the database server remains in read-only mode.

Considerations After High-Availability Data-Replication Failure

Consider the following two issues when a high-availability data-replication 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 27-20](#).

- How the database servers should react to the failure

Which administrative actions to take after a high-availability data-replication 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 27-19](#) and [“Actions to Take If the Primary Database Server Fails” on page 27-19](#).

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 in on-line mode.

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 27-20](#). 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 28-33](#).

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 high-availability data-replication connection to be restored very soon.
- Use *manual switchover* to change the database server mode to standard.

The following sections describe manual switchover.

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 on-line 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 Type of the Secondary Database Server” on page 28-23](#).

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 28-34.

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 28-34.

Redirection and Connectivity for Data-Replication Clients

To connect to the database servers in a high-availability data-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 [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 high-availability data-replication pair, but you can automate this action from within the application, as described in [“Handling Redirection Within an Application”](#) on page 27-28.

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.

Automatic Redirection 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*

The **DBPATH** redirection method relies on the fact that 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 high-availability data-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 high-availability data-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 [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.

Administrator-Controlled Redirection with the Connectivity Information

This section explains the steps in redirecting clients with the connectivity information and the connectivity strategy that supports this method. As the following table shows, the location of the connectivity information depends on the platform.

Operating System	Location of Connectivity Information
UNIX	The INFORMIXSQLHOSTS environment variable specifies the full pathname and filename of the connection information. The default location is \$INFORMIXDIR/etc/sqlhosts . For more information about INFORMIXSQLHOSTS , see the Informix Guide to SQL: Reference .
Windows NT	The connectivity information is in a key in the Windows registry called HKEY_LOCAL_MACHINE\SOFTWARE\Informix\SQLHOSTS .

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 high-availability data-replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in “[What the Administrator Needs to Do](#),” the fields of the unavailable database server (except for the **dbservername** field) are changed to point to a definition of the remaining database server in the high-availability data-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 if 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*

What the Administrator Needs to Do

To use the connectivity information to redirect clients, administrators must perform the following two steps:

1. Change the connectivity information for the clients.
2. Change other connectivity files, if necessary.

The following sections describe these steps. For information on the connectivity information, refer to [Chapter 5, “Client/Server Communications.”](#)

Changing the Connectivity Information

On the client computer, edit the connectivity information. Make the following changes:

- Comment out the entry for the failed database server.
- 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.

Figure 27-7 on page 27-26 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.

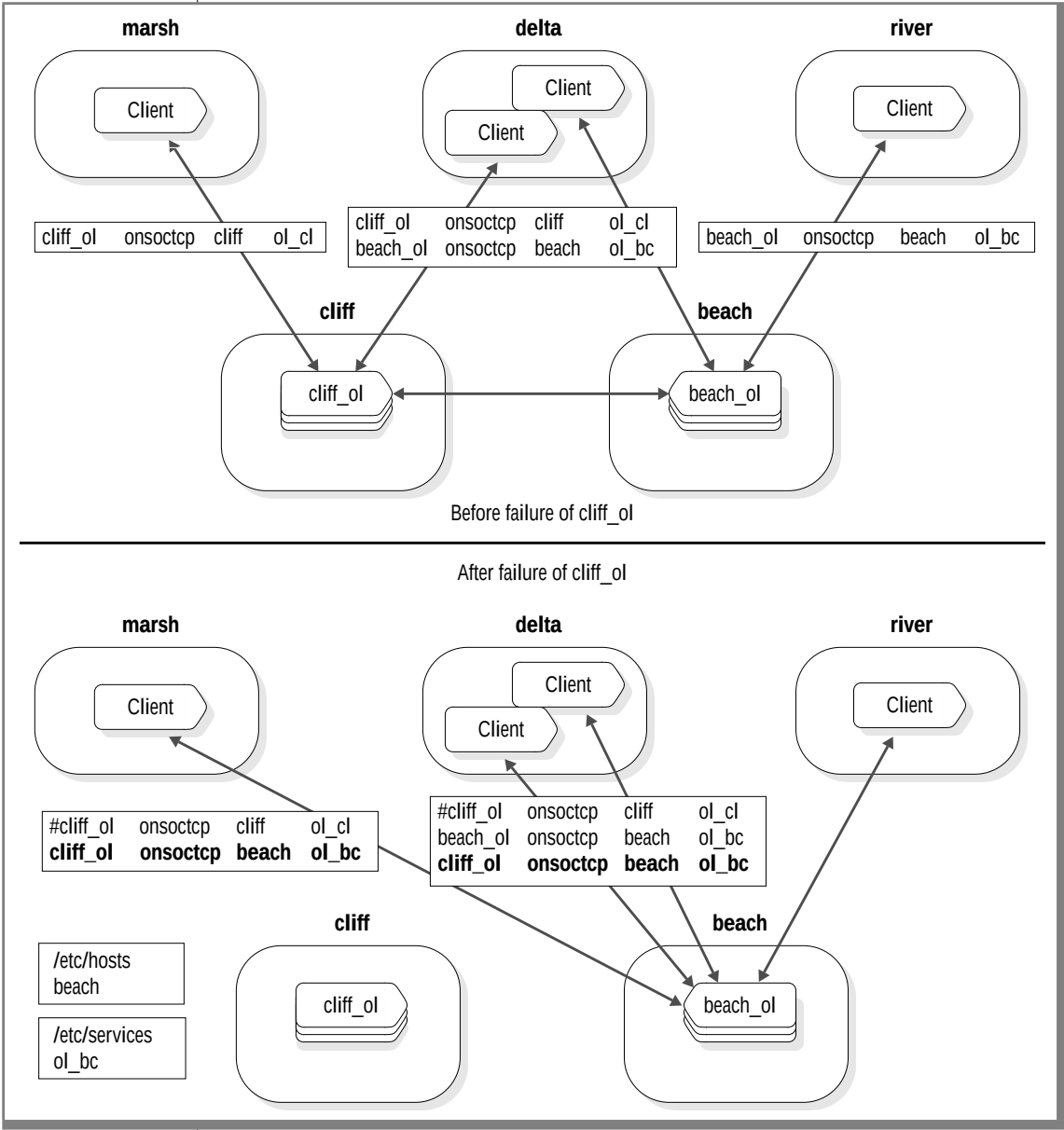
Changing Other Connectivity Files

You also must ensure that the following statements are true on the client computer before that client can reconnect to the other database server:

- The **/etc/hosts** file on UNIX or **hosts** file on Windows NT has an entry for the **hostname** of the computer that is running the database server to which you are redirecting clients.
- The **/etc/services** file on UNIX or **services** file on Windows NT has an entry for the **servicename** of the database server to which you are redirecting clients.

Figure 27-7

Connectivity Values Before and After a Failure of the cliff_ol Database Server



What the User Needs to Do

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.

User-Controlled Redirection 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 high-availability data-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 attend to changing 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:

1. Quit their applications.
2. Change their **INFORMIXSERVER** environment variable to hold the name of the other database server in the high-availability data-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 a high-availability data-replication 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 27-8 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.

Figure 27-8
*Example of a
 CONNECT Loop for
 DBPATH Redirection
 Mechanism*

```

/* 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);
}

```

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).

Comparison of Different Redirection Mechanisms

Figure 27-9 summarizes the differences among the three redirection mechanisms.

Figure 27-9
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 are redirected.	Individual clients are redirected.	All clients that use a given database server are redirected.	Individual clients are redirected.	Individual clients are redirected.
Are changes to environment variables required?	No		No		Yes

Designing High-Availability Data-Replication 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_btsend** thread. The application thread that is executing is free to continue processing. The **dr_btsend** 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 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 (which includes all non-temporary dbspaces on the secondary database server) are not allowed. For example, the following statements produce errors:

- ALTER FRAGMENT
- ALTER INDEX
- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE PROCEDURE
- CREATE PROCEDURE FROM
- CREATE ROLE
- CREATE SCHEMA

- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DELETE
- DROP DATABASE
- DROP INDEX
- DROP PROCEDURE
- DROP ROLE
- DROP SYNONYM
- DROP TABLE
- DROP TRIGGER
- DROP VIEW
- GRANT
- GRANT FRAGMENT
- INSERT
- LOAD
- RENAME COLUMN
- RENAME DATABASE
- RENAME TABLE
- REVOKE
- REVOKE FRAGMENT
- UNLOAD
- UPDATE
- UPDATE STATISTICS

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.
- Conditionalize all updating statements.

To conditionalize statements that perform an update, 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.

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 14-22](#) 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 15-13](#).
- 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

Planning for High-Availability Data Replication	28-4
Configuring a System for High-Availability Data Replication	28-5
Meeting Hardware and Operating-System Requirements	28-5
Meeting Database and Data Requirements	28-6
Meeting Database Server Configuration Requirements	28-6
Database Server Version	28-7
Dbospace and Chunk Configuration	28-7
Mirroring	28-7
Physical-Log Configuration	28-8
Dbospace and Logical-Log Tape Backup Devices	28-8
Logical-Log Configuration	28-8
Shared-Memory Configuration	28-9
High-Availability Data-Replication Parameters	28-9
Configuring High-Availability Data-Replication Connectivity.	28-9
Starting High-Availability Data Replication for the First Time	28-11
Performing Basic Administration Tasks.	28-16
Changing Database Server Configuration Parameters	28-16
Backing Up Dbospaces and Logical-Log Files.	28-17
Changing the Logging Status of Databases	28-18
Adding and Dropping Chunks, Dbospaces, and Blobspaces.	28-18
Using and Changing Mirroring of Chunks	28-18
Managing the Physical Log.	28-19
Managing the Logical Log	28-20
Managing Virtual Processors	28-20
Managing Shared Memory.	28-20

Changing the Database Server Mode	28-21
Changing the Database Server Type	28-22
Changing the Type of the Primary Database Server	28-23
Changing the Type of the Secondary Database Server.	28-23
Monitoring High-Availability Data-Replication Status	28-24
Using Command-Line Utilities	28-24
Using SMI Tables	28-26
Using ON-Monitor.	28-26
Restoring Data After Media Failure Occurs	28-26
Restoring After a Media Failure on the Primary Database Server.	28-27
Restoring After a Media Failure on the Secondary Database Server	28-28
Restarting High-Availability Data Replication After a Failure	28-29
Restarting After Critical Data Is Damaged	28-30
Critical Media Failure on the Primary Database Server	28-30
Critical Media Failure on the Secondary Database Server	28-31
Critical Media Failure on Both Database Servers	28-32
Restarting If Critical Data Is Not Damaged	28-32
Restarting After a Network Failure.	28-33
Restarting If the Secondary Database Server Fails.	28-33
Restarting If the Primary Database Server Fails	28-34

In This Chapter

This chapter describes how to use high-availability data replication for Dynamic Server. If you plan to use high-availability data replication, read this entire chapter first. If you plan to use Informix Enterprise Replication, see the [Guide to Informix Enterprise Replication](#). This chapter covers the following topics:

- Planning for high-availability data replication
- Configuring a system for high-availability data replication
- Starting high-availability data replication for the first time
- Performing basic administration tasks
- Changing the database server mode
- Changing the database server type
- Monitoring high-availability data-replication status
- Restoring data after a media failure
- Restarting high-availability data replication after a failure

[Chapter 27, “High-Availability Data Replication,”](#) explains what high-availability data replication is, how it works, and how to design client applications for a high-availability data-replication environment.

Planning for High-Availability Data Replication

Before you start setting up computers and database servers to use high-availability data replication, 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 18, “Managing Database-Logging Status.”](#)
- Develop client applications to make use of both database servers in the high-availability data-replication pair. For a discussion of design considerations, refer to [“Redirection and Connectivity for Data-Replication Clients” on page 27-20](#) and [“Designing Clients to Use the Secondary Database Server” on page 27-32](#).
- Create a schedule for starting high-availability data replication for the first time.
- Design a dbspace 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 high-availability data replication after a failure. Read [“Redirection and Connectivity for Data-Replication Clients” on page 27-20](#).

Configuring a System for High-Availability Data Replication

To configure your system for high-availability data replication, you must take the following actions:

- Meet hardware and operating-system requirements.
- Meet database and data requirements.
- Meet database server configuration requirements.
- Configure high-availability data-replication connectivity.

Each of these topics is explained in this section.

Meeting Hardware and Operating-System Requirements

For a high-availability data-replication 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 non-temporary 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 a high-availability data-replication database server pair to function, you must meet the following database and data requirements:

- 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 high-availability data replication, see either [“Turning On Transaction Logging with ON-Archive” on page 18-7](#) or [“Turning On Transaction Logging with ontape” on page 18-9](#).

- 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 high-availability data-replication processing. However, simple-large-object data within dbspaces is replicated.

Meeting Database Server Configuration Requirements

For a high-availability data-replication database server pair to function, you must meet the following database server configuration requirements.

To meet these requirements, 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 3-1](#). 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*.

UNIX

Database Server Version

The versions of the database server on the primary and secondary database servers must be identical.

Dbospace 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.

The configuration must contain at least one temporary dbspace. See [“Use of Temporary Dbspaces for Sorting and Temporary Tables”](#) on page 27-35.

You can use symbolic links for the chunk pathnames, as explained in [“Allocating Raw Disk Space on UNIX”](#) on page 15-10. ♦

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME
- ROOTOFFSET
- 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

DbSPACE and Logical-Log Tape Backup Devices

You can specify different tape devices for the primary and secondary database servers.

The tape size and tape block size for the dbSPACE and logical-log tape backup devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK
- LTAPESIZE
- TAPEBLK
- TAPESIZE

Logical-Log Configuration

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

Shared-Memory Configuration

Set all the shared-memory configuration parameters (SHMADD, SHMTOTAL, SHMVIRTSIZE) to the same values on the two database servers.

High-Availability Data-Replication Parameters

The following parameters are specific to high-availability data replication and must be set to the same value on both database servers in the high-availability data-replication pair:

- DRINTERVAL
- DRLOSTFOUND
- DRTIMEOUT

Configuring High-Availability Data-Replication Connectivity

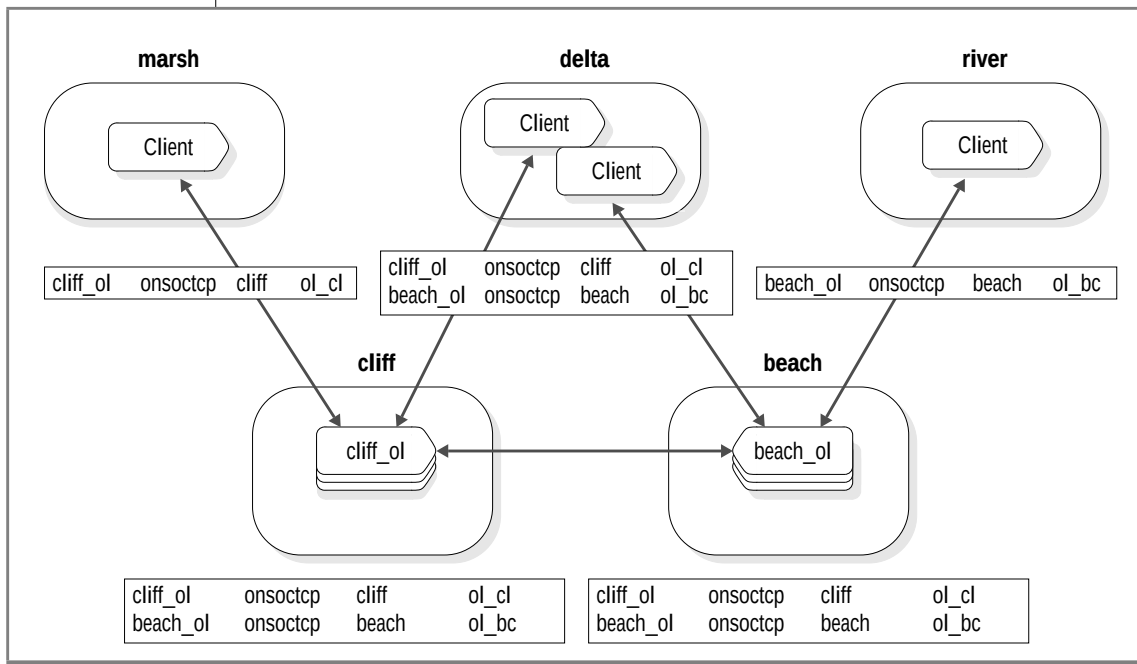
For a high-availability data-replication database server pair to function, the database servers in the high-availability data-replication pair must be able to establish a connection with one another. To satisfy this requirement, the connectivity information on each of the computers that is running the database server in a high-availability data-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

Figure 28-1 shows a sample high-availability data-replication configuration and sample connectivity values necessary for high-availability data replication.

Figure 28-1

Sample Connectivity Values for Database Servers in a High-Availability Data-Replication Pair



In addition to connectivity information, the computers that are running Dynamic Server in a high-availability data-replication pair must have entries for the other computer and service in their **/etc/hosts** and **/etc/services** files on UNIX or their **hosts** and **services** files on Windows NT.

Starting High-Availability Data Replication for the First Time

After you complete the high-availability data-replication configuration, you are ready to start high-availability data replication. This section describes the necessary steps for starting high-availability data replication.

Suppose you want to start high-availability data replication on two database servers, **ServerA** and **ServerB**. The procedure for starting high-availability data replication, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps.

[Figure 28-2 on page 28-13](#) lists the commands required to perform each step. You can use ON-Archive, **ontape**, or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure, however. [Figure 28-2](#) also shows messages sent to the message log.

To start high-availability data replication

1. Create a level-0 dbspace backup of **ServerA**.

If data replication is being restarted, the primary database server should be in standard mode, not primary mode.

2. 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 high-availability data-replication connection with the other database server in the high-availability data-replication pair and to start high-availability data-replication 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 on-line and is not set to type secondary, so the high-availability data-replication connection is not established.

3. 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.

4. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server. **ServerB** tries to establish a high-availability data-replication connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

Before high-availability data replication 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 5.



Important: You must complete steps 3 and 4 together. If you need to shut down and restart the secondary database server after step 3, you must redo step 3.

5. 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 28-2
Steps to Start High-Availability Data Replication for the First Time

Step	On the Primary	On the Secondary
1	ON-Archive command Onarchive> ARCHIVE/DBSPACESET=* 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.	
2	onmode command %onmode -d primary sec_name Messages to message log DR: new type = primary, secondary server name = sec_name DR: trying to connect to secondary server DR: Cannot connect to secondary server	

(1 of 4)

Starting High-Availability Data Replication for the First Time

Step	On the Primary	On the Secondary
3		ON-Archive command ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=rid/TAPE= (primary:/dev/remotedrive) 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 Informix Database Server Initialized -- Shared Memory Initialized Recovery Mode Physical restore of rootdbs started. Physical restore of rootdbs completed.

(2 of 4)

Step	On the Primary	On the Secondary
4	Messages to message log DR: Primary server connected DR: Primary server operational	onmode command % onmode -d secondary <i>prim_name</i> Messages to message log DR: new type = secondary, primary server name = <i>prim_name</i> 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). Messages to message log DR: Trying to connect to primary server DR: Secondary server connected DR: Failure recovery from disk in process. Logical Recovery allocating n worker threads ('OFF_RECVR_THREADS'). Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xxxxxx DR: Secondary server operational

(3 of 4)

Step	On the Primary	On the Secondary
5	Messages to message log DR: Primary server connected DR: Primary server operational	ON-Archive command ONDATARTR> RETRIEVE/LOGFILE/TAPE= (primary:/dev/remotedevice) ontape command % ontape -l ON-Bar command onbar -l Messages to message log DR: Secondary server connected DR: Failure recovery from disk in process. Logical Recovery allocating n worker threads ('OFF_RECVR_THREADS'). Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xnnnnn DR: Secondary server operational

(4 of 4)

Performing Basic Administration Tasks

This section contains instructions on how to perform basic Dynamic Server administration tasks once your system is running high-availability data replication.

Changing Database Server Configuration Parameters

Some of the Dynamic Server configuration parameters must be set to the same value on both database servers in the high-availability data-replication pair (as listed under [“Meeting Database Server Configuration Requirements” on page 28-6](#)). 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 off-line with the **onmode -k** option.
2. Change the parameters on each database server.
3. Bring each database server back on-line. Start with the last database server that you brought off-line. For example, if you brought the secondary database server off-line last, bring the secondary database server on-line first. [Figure 28-1 on page 28-10](#) and [Figure 28-2 on page 28-13](#) list the procedures for bringing the primary and secondary database servers back on-line.

If the configuration parameter does not need to have the same value on each database server in the high-availability data-replication pair, you can change the value on the primary or secondary database server individually.

Backing Up Dbspaces and Logical-Log Files

When you use high-availability data replication, you must back up logical-log files and storage spaces, just as you would with Dynamic Server. You need to perform storage-space and logical-log backups only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.

You must use the same backup and restore tool on both database servers.

The block size and tape size used (for both storage-space backups and logical-log backups) must be identical on the primary and secondary database servers.

Changing the Logging Status of Databases

You cannot add transaction logging to databases on the primary database server while you are using high-availability data replication. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

If you must add logging to a database, you must turn high-availability data replication off, add logging, and perform a level-0 backup on the primary database server and restore on the secondary database server. This procedure is described in [“Starting High-Availability Data Replication for the First Time” on page 28-11](#). After you add logging to 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

Adding and Dropping Chunks, Dbspaces, and Blobspaces

You can perform disk-layout operations, such as adding or dropping chunks, dbspaces, blobspaces, and sbspaces, 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 high-availability data-replication pair remains consistent.

Because 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.

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 high-availability data-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 high-availability data-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 15-6](#).

You can perform disk-layout operations only from the primary database server. 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 25-12](#).

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 25-13](#).

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 high-availability data replication.

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 PHYFILE parameters in the secondary ONCONFIG file are not updated. You must change these parameters manually by editing the ONCONFIG file. For the procedure to follow for making this change, see [“Changing Database Server Configuration Parameters” on page 28-16](#).

For information on changing the size and location of the physical log, refer to [Chapter 22, “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 20, “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 to follow for making this change, see [“Changing Database Server Configuration Parameters” on page 28-16.](#)

Changing the Database Server Mode

The effects of changing the mode of a database server in a high-availability data-replication pair differ depending on whether you are changing the mode of the primary or the secondary database server.

[Figure 28-3](#) summarizes the effects of changing the mode of the primary database server.

Figure 28-3
Mode Changes on the Primary Database Server

On the Primary	On the Secondary	To Restart High-Availability Data Replication
Any mode off-line (onmode -k)	Secondary receives errors. High-availability data replication 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 off-line: ■ “The Secondary Database Server Was Not Changed to a Standard Database Server” on page 28-34 ■ “The Secondary Database Server Is Changed to a Standard Database Server” on page 28-34
On-line or quiescent (onmode-s / onmode-u)	Secondary does not receive errors. High-availability data replication remains on. Mode remains read-only.	Use onmode -m on the primary.

Figure 28-4 summarizes the effects of changing the mode of the secondary database server.

Figure 28-4
Mode Changes on the Secondary Database Server

On the Secondary	On the Primary	To Restart High-Availability Data Replication
Read-only off-line (onmode -k)	Primary receives errors. High-availability data replication 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 28-33.

Changing the Database Server Type

You might want to stop the high-availability data-replication process by changing the type of the database server to *standard*. The effects of this change are different from changing the mode of a database server (described in [“Changing the Database Server Mode”](#) on page 28-21). When you take the now standard database server off-line and bring it back on-line, it does not attempt to connect to the other database server in the high-availability data-replication pair.

To change the database server type, use the **onmode** utility. For information about **onmode**, see the utilities chapter of the *Administrator’s Reference*.

You can change the type of either the primary or the secondary database server. When you change the database server type to *standard*, the type of the other database server in the high-availability data-replication pair does not change, but high-availability data replication is turned off.

Changing the Type of the Primary Database Server

The primary database server can be in on-line mode when you change its type to standard.

Execute the following command from the operating-system prompt of the computer that is running the primary database server:

```
% onmode -d standard
```

This command stops high-availability data replication and leaves the database server in on-line mode. The secondary server remains in read-only mode and cannot accept updates from clients (because its type is still secondary). High-availability data replication is turned off on the secondary database server.

To change the database server back to type primary and restart high-availability data replication, execute the following command:

```
% onmode -d primary secondary
```

Changing the Type of the Secondary Database Server

Execute the following command from the operating-system prompt of the computer that is running the secondary database server:

```
% onmode -d standard
```

Once you change the secondary database server to a standard database server, applications can update the data managed by that database server. You should change the secondary database server to a standard database server only if the primary database server is not set up.

If you change the secondary database server to a standard database server when the primary database server is up, to restart high-availability data replication, you must follow the entire procedure in [“Starting High-Availability Data Replication for the First Time”](#) on page 28-11.

Monitoring High-Availability Data-Replication Status

Monitor the high-availability data-replication 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 high -availability data replication is on
- The values of the high-availability data-replication 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 high-availability data-replication pair, and in on-line mode:

```
Informix Dynamic Server 2000 Version 9.20.UC1      -- On-Line(Prim) -- Up 45:08:57
```

This example shows a database server that is the secondary database server in a high-availability data-replication pair, and in read-only mode.

```
Informix Dynamic Server 2000 Version 9.20.UC1      -- Read-Only (Sec) -- Up 45:08:57
```

The following example shows a header for a database server that is not involved in high-availability data replication. The type for this database server is standard.

```
Informix Dynamic Server 2000 Version 9.20.UC1      -- On-Line -- Up 20:10:57
```

onstat -g dri

To obtain full high-availability data-replication monitoring information, execute the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The high-availability data-replication state (on or off)

- The paired database server
- The last high-availability data-replication checkpoint
- The values of the high-availability data-replication configuration parameters

Example output is shown in [Figure 28-5](#). This example shows a primary database server, paired with a secondary database server that has the DBSERVERNAME of **beach_ol**. High-availability data replication 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 28-5
onstat -g dri Output

oncheck -pr

If your database server is running high-availability data replication, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that high-availability data replication uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in [Figure 28-6](#).

```
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 28-6
*oncheck -pr
PAGE_1ARCH
Output for Database
Server Running
High-Availability
Data Replication*

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	High-availability data-replication server type
state	High-availability data-replication server state
name	Database server name
intvl	High-availability data-replication buffer flush interval
timeout	Network timeout
lostfound	High-availability data-replication lost+found pathname

Using ON-Monitor

Use the Status menu, data-Replication option, to see information about high-availability data replication. This option displays the same information as the **onstat -g dri** command-line 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 28-7 on page 28-28](#) summarizes the various scenarios for restoring data if the primary database server suffers media failure. The following issues are relevant:

1. If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.
2. 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 28-30](#).

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 dbspace backup and next a logical restore of all logical-log records written since that dbspace backup. For more information on performing a warm restore with ON-Bar, see your *Backup and Restore Guide*. For more information on performing a warm restore with ON-Archive, see your [Archive and Backup Guide](#). You must back up all logical-log files before you perform the warm restore.

Figure 28-7
Scenarios for Media Failure on the Primary Database Server

High-Availability Data-Replication 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 28-30.
Primary	Yes	Yes	Primary database server remains on-line. Follow the procedures in “Recovering a Mirrored Chunk” on page 25-12.
Primary	No	No	Primary database server remains on-line. 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 on-line. Follow the procedures in “Recovering a Mirrored Chunk” on page 25-12.

Restoring After a Media Failure on the Secondary Database Server

[Figure 28-8](#) summarizes the various scenarios for restoring data if the secondary database server suffers media failure. The following issues are relevant:

1. If chunks are mirrored, you can perform recovery just as you would for a standard database server that uses mirroring.
2. In cases where the chunks are not mirrored, the secondary database server fails if the disk contains critical media but remains on-line 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 28-30.](#)) 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 28-8
Scenarios for Media Failure on the Secondary Database Server

High-Availability Data-Replication Server	Critical Media	Chunks Mirrored	Effect of Failure
Secondary	Yes	No	Secondary database server fails. Primary database server receives errors. High-availability data replication is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 28-30
Secondary	Yes	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 25-12.
Secondary	No	No	Secondary database server remains on-line in read-only mode. Primary database server receives errors. High-availability data replication is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 28-30.
Secondary	No	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 25-12.

Restarting High-Availability Data Replication After a Failure

[“High-Availability Data-Replication Failures Defined”](#) on page 27-16 discusses the various types of high-availability data-replication failure. The procedure that you must follow to restart high-availability data replication 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 high-availability data replication 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

To restart high-availability data replication after the primary database server suffers a critical media failure, perform the following steps. [Figure 28-9](#) lists the commands required to perform this procedure:

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 high-availability data replication. 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, high-availability data replication does not start until you perform step 5.
4. The secondary database server is now in fast recovery. Do a SELECT from the secondary database server.
5. 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 28-9
Steps for Restarting High-Availability Data Replication 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 ON-Archive command ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST= rid/TAPE=(primary:/dev/remotedrive)	
3	onmode command % onmode -d primary <i>sec_name</i>	
4	ontape command % ontape -l ON-Archive command ONDATARTR> RETRIEVE/LOGFILE/TAPE= (secondary:/dev/remotedevice)	

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 High-Availability Data Replication for the First Time”](#) on page 28-11.

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the computers that are running database servers in a high-availability data-replication pair experience a failure that damages the root dbspace, the dbspace that contains logical-log files or the physical log, perform the following tasks to restart high-availability data replication:

1. Restore one database server—it does not matter which one—from dbspace and logical-log backup tapes.
2. After you restore one database server, treat the other failed database server as if it has no data on the disks and you are starting high-availability data replication for the first time. (See [“Starting High-Availability Data Replication for the First Time” on page 28-11.](#)) Use the functioning database server with the intact disk(s) 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 high-availability data replication, 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 on-line mode, and the secondary database server is in read-only mode. High-availability data replication is turned off on both database servers (state = off). When the connection is reestablished, you can restart high-availability data replication by issuing **onmode -d secondary primary_name** on the secondary database server. Restarting high-availability data replication 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 high-availability data replication after a failure of the secondary database server, complete the steps in [Figure 28-10](#). 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 28-10
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 on-line mode.	% oninit If you receive the following message in the message log, continue with step 2: DR: Start Failure recovery from tape
2		ON-Archive command Onarchive> CATALOG/VSET=remote_logs/VOLUME=volnum/SID=sysid Onarchive>RETRIEVE/LOGFILE/VSET=remote_logs ontape command % ontape -l

Restarting If the Primary Database Server Fails

The following sections describe how to restart high-availability data replication 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 high-availability data replication 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 on-line using **oninit**.

The Secondary Database Server Is Changed to a Standard Database Server

If you need to restart high-availability data replication 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 28-11](#).

Figure 28-11
Steps to Restart If You Changed the Secondary Database Server to Standard

Step	On the Primary Database Server	On the Secondary Database Server
1		<pre>% onmode -s</pre> 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 27-20.
2		<pre>% onmode -d secondary prim_name</pre>

(1 of 2)

Step	On the Primary Database Server	On the Secondary Database Server
3	% oninit If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the oninit command. 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). For ontape users: 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. For ON-Archive users: In the next step, be sure to use a vset with the device type defined to be a device on the secondary database server.	
4	If you are prompted to recover logical-log records from tape, perform this step. ON-Archive command <pre>Onarchive> CATALOG/VSET=remote_logs/VOLUME= volnum/SID=sysid Onarchive>RETRIEVE/LOGFILE/VSET= remote_logs</pre> ontape command <pre>% ontape -l</pre>	

(2 of 2)

Consistency Checking

Performing Periodic Consistency Checking	29-3
Verifying Consistency	29-4
Validating Data Pages	29-4
Validating Extents	29-5
Validating Indexes	29-5
Validating Logical Logs	29-5
Validating Metadata	29-5
Validating Reserved Pages	29-5
Validating System Catalog Tables	29-6
Monitoring for Data Inconsistency	29-6
Assertion Failures in the Message Log and Dump Files	29-6
Validating Table and Tablespace Data	29-7
Retaining Consistent Level-0 Backups	29-8
Dealing with Corruption	29-8
Symptoms of Corruption	29-9
Fixing Index Corruption	29-9
I/O Errors on a Chunk	29-9
Collecting Diagnostic Information	29-11

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 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

Performing Periodic Consistency Checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, Informix recommends 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. Informix recommends that you perform this check just before you create a level-0 dbspace backup.

Run the commands shown in [Figure 29-1](#) as part of the consistency check.

Figure 29-1
Checking Data Consistency

Type of Validation	Command
Data	oncheck -cD <i>dbname</i>
Extents	oncheck -ce
Indexes	oncheck -cI <i>dbname</i>
Metadata	oncheck -cs
Reserved pages	oncheck -cr
System catalog tables	oncheck -cc

You can run each of these commands while the database server is in on-line 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 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 [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 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.

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 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. 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.
```

This message indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**.

However, if you receive an error message when you validate system catalog tables, the situation is quite different. Contact Informix Technical Support immediately.

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*.)

Assertion Failures in the Message Log and Dump Files

Figure 29-2 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 29-2
*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 tablespace 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 be 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 [“Verifying Consistency” on page 29-4](#) 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. Informix recommends 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.

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 [“Monitoring for Data Inconsistency” on page 29-6](#).
- 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 on-line 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 on-line 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.

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 29-3 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 fstdbs
  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 29-3
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 25-12.

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 off-line 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 Informix Technical Support whenever one occurs. The diagnostic information collected is intended for the use of 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 Informix Technical Support indicates that you can do so. Send a fax or email with the **af.xxx** file to Informix Technical Support. This 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.

UNIX

Distributed Data

Chapter 30 Multiphase Commit Protocols

Chapter 31 Recovering Manually from Failed Two-Phase Commit

Section VII



Multiphase Commit Protocols

Two-Phase Commit Protocol	30-3
When the Two-Phase Commit Protocol Is Used.	30-3
Goals for the Two-Phase Commit Protocol	30-4
Two-Phase Commit Concepts	30-5
Phases of the Two-Phase Commit Protocol	30-6
Precommit Phase	30-6
Postdecision Phase	30-7
Examples of Two-Phase Commit Transactions	30-8
How the Two-Phase Commit Protocol Handles Failures.	30-10
Types of Failures That Automatic Recovery Handles	30-10
Administrator's Role in Automatic Recovery	30-10
Automatic-Recovery Mechanisms for Coordinator Failure	30-10
Automatic-Recovery Mechanisms for Participant Failure	30-14
Presumed-Abort Optimization	30-17
How Presumed-Abort Optimization Affects Automatic Recovery	30-18
Why Optimization Is Realized	30-18
Independent Actions	30-18
Situations That Initiate Independent Action	30-19
Possible Results of Independent Action	30-19
Independent Actions That Allow Successful Completion of Transaction	30-20
Independent Actions That Result in an Error Condition	30-20
Independent Actions That Result in Heuristic Decisions	30-21
The Heuristic Rollback Scenario	30-22
Conditions That Result in a Heuristic Rollback	30-22
Results of a Heuristic Rollback	30-23

The Heuristic End-Transaction Scenario	30-26
When to Perform a Heuristic End Transaction	30-26
How to Use onmode -Z	30-27
Action When the Transaction Is Ended Heuristically	30-28
Tracking a Global Transaction	30-28
Two-Phase Commit Protocol Errors	30-29
Two-Phase Commit and Logical-Log Records	30-30
Logical-Log Records When the Transaction Commits	30-31
Logical-Log Records Written During a Heuristic Rollback	30-33
Logical-Log Records Written After a Heuristic End Transaction	30-35
Configuration Parameters Used in Two-Phase Commits	30-37
Function of the DEADLOCK_TIMEOUT Parameter	30-37
Function of the TXTIMEOUT Parameter	30-37
Heterogeneous Commit Protocol	30-38
Gateways That Can Participate in a Heterogeneous Commit Transaction	30-39
Enabling and Disabling Heterogeneous Commit	30-40
How Heterogeneous Commit Works	30-41
Precommit Phase	30-41
Gateway Commit Phase	30-41
Heterogeneous Commit Optimization	30-42
Implications of a Failed Heterogeneous Commit	30-43
Database Server Coordinator Failure	30-43
Participant Failure	30-44
Interpretation of Heterogeneous Commit Error Messages	30-45

In This Chapter

A multiphase commit protocol is a mechanism that Dynamic Server uses to process transactions that span multiple database servers. Understanding how multiphase commit protocols work becomes important only when a deviation from one of these protocols occurs. The database server supports two multiphase commit protocols: two-phase commit protocol and heterogeneous commit protocol.

For information on recovering manually from a failed two-phase commit, see [Chapter 31, “Recovering Manually from Failed Two-Phase Commit.”](#)

Two-Phase Commit Protocol

The two-phase commit protocol governs the order in which a two-phase commit transaction is performed and provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction.

When the Two-Phase Commit Protocol Is Used

A database server automatically uses the two-phase commit protocol for any transaction that performs modifications to data on more than one database server.

Goals for the Two-Phase Commit Protocol

For example, consider the configuration shown in [Figure 30-1](#), which includes three database servers.

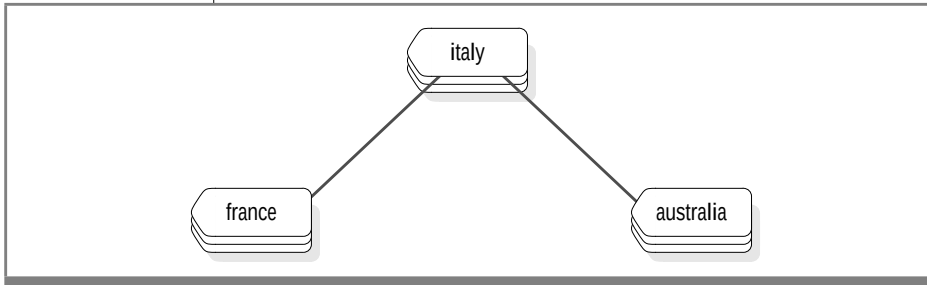


Figure 30-1
*Connected
Database Servers*

If you execute the commands shown in [Figure 30-2](#), 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 30-2
*Example of a
Distributed
Transaction*

The database server automatically uses the two-phase commit protocol for this transaction.

Goals for the Two-Phase Commit Protocol

Distributed transactions that include multiserver modifications use the two-phase commit protocol to achieve the following two goals:

- Ensure that all participating database servers receive the same instruction, either to commit or to roll back a transaction
- Ensure that all participating database servers implement the same action (either a commit or a rollback), regardless of local or network failures during the protocol

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. The database server uses the two-phase commit protocol to coordinate the work performed at multiple database servers on behalf of a single transaction.

Two-Phase Commit Concepts

Every transaction that uses the two-phase commit protocol has a *coordinator* and one or more *participants*:

- Coordinator

The two-phase commit protocol always assigns the role of coordinator to the *current* database server. In other words, the database server that is managing the current database when the transaction is processed is the coordinator. In the sample transaction in [Figure 30-2 on page 30-4](#), 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
```

The role of coordinator cannot change during a single transaction. The coordinator for a distributed transaction is displayed by the **onstat -x** option. For an example of **onstat -x** output, refer to the monitoring transactions section in your [Performance Guide](#).

In the two-phase commit protocol, the transaction that is under the direction of the coordinator is called the *global transaction*.

- Participants

The database servers that perform operations under the direction of the coordinator are the participants. In [Figure 30-1](#), the participants are **france** and **australia**. The work that each participant database server performs is a *piece of work* associated with the global transaction. In this example, the coordinator database server, **italy**, also functions as a participant because it is also doing a piece of work, which is the update.

The two-phase commit protocol relies on two kinds of communication, messages and logical-log records:

- **Messages**

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**

Logical-log records of the transaction are kept on stable storage (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 about the logical-log records that are written during two-phase commit protocol, refer to [“Two-Phase Commit and Logical-Log Records” on page 30-30](#).

Phases of the Two-Phase Commit Protocol

In a two-phase commit transaction, all the instructions are first sent to all the participants to perform data modifications (for example, inserts). After the coordinator sends these instructions, it 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:** If the data modifications satisfy all deferred constraints, the participants return messages to the coordinator indicating that their piece of work can be committed. If a modification does not satisfy a constraint, the participant returns a negative response to the coordinator.

3. **Coordinator:** The coordinator determines whether to commit or roll back the transaction. If at least one participant is unable to perform the modifications, the coordinator makes the decision to roll back the 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 send 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.

Examples of Two-Phase Commit Transactions

Figure 30-3 represents a two-phase commit protocol that results in a committed transaction.

The postdecision phase begins at the instant when the coordinator records its decision to commit or roll back. In this case, phase two begins when the coordinator writes the commit work logical-log record to disk.

Figure 30-3

Two-Phase Commit Protocol That Results in a Committed Transaction

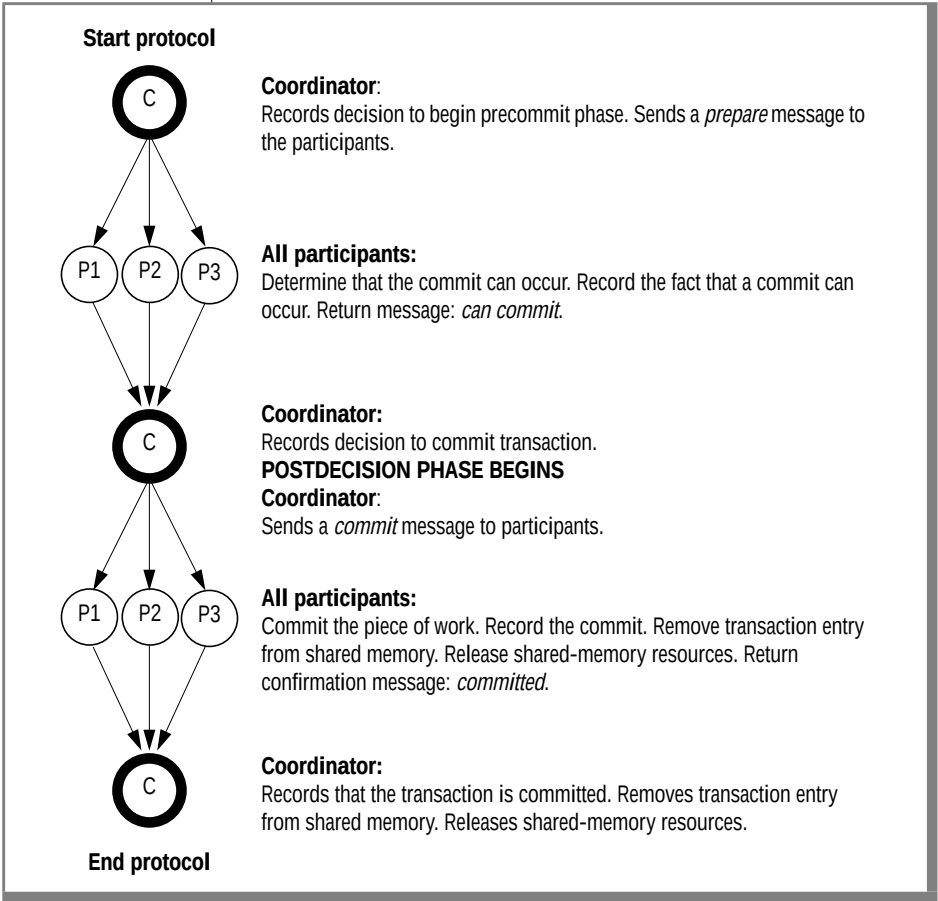
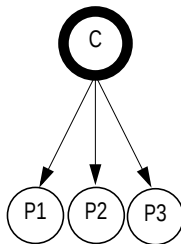


Figure 30-4 represents a two-phase commit that results in a rolled-back transaction. For some reason, probably a deferred constraint violation, the P1 participant determines that the commit cannot occur and returns a `cannot commit` message to the coordinator.

The participants do not send a confirmation to the coordinator when the piece of work is rolled back. The coordinator does not record a completed transaction. In this example, the postdecision phase begins when the coordinator writes the rollback-work logical-log record to disk.

Start protocol



Coordinator:

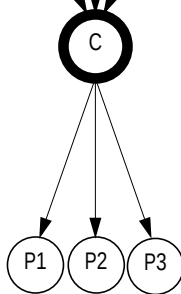
Records decision to begin precommit phase. Sends a *prepare* message to the participants.

Participant 1:

Determines that the commit cannot occur. Returns message: *cannot commit*. Rolls back piece of work.

Participants 2 and 3:

Determine that the commit can occur. Record the fact that a commit can occur. Return message: *can commit*.



Coordinator:

Receives *cannot commit* response. Records decision to roll back transaction.

PHASE 2 BEGINS

Coordinator:

Rolls back work on behalf of transaction. Records rollback. Removes transaction entry from shared memory. Releases shared-memory resources. Sends a *roll back* message to participants.

Participants 2 and 3:

Roll back piece of work. Remove transaction entry from shared-memory transaction table. Release shared-memory resources.

End protocol

Figure 30-4

A Successful Two-Phase Commit in Which Transaction Rolls Back

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 on-line after a system or network failure.

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 30-17.](#))

The coordinator must initiate recovery if the coordinating thread fails after a decision to commit the transaction but before the two-phase commit protocol is complete.



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.

Coordinator Fails Before Decision to Commit Transaction

If the coordinator fails before it decides whether to commit the transaction, the recovery mechanism is as follows. First, the system administrator must bring the coordinator back on-line. Next, the fast-recovery mechanism rolls back the global transaction on the coordinator. Meanwhile, the participant waits for the time period specified by the configuration parameter `TXTIMEOUT` for a message from the coordinator to either commit or roll back its piece of work. After waiting the specified time, the participant attempts to fork a new thread on the coordinator to determine the transaction status. (The participant will not be able to fork this new thread until the coordinator is back on-line). Because the coordinator rolled back the global transaction, no transaction is in shared memory. The presumed-abort optimization is in effect, and the participant rolls back its piece of work.

Coordinator Fails After Decision to Roll Back Transaction

If the coordinator fails after deciding to roll back the transaction, but before completing the two-phase commit protocol, the recovery mechanism is as follows. First, the system administrator must bring the coordinator back on-line. The global transaction at the coordinator has already been removed from memory, so the coordinator takes no further action. Meanwhile, the participant waits for the time period specified by the configuration parameter `TXTIMEOUT` for a message from the coordinator to either commit or roll back its piece of work. After waiting the specified time, the participant attempts to fork a new thread on the coordinator to determine the transaction status. (The participant will not be able to fork this new thread until the coordinator is back on-line). Because the coordinator rolled back the global transaction, no transaction is in shared memory. The presumed-abort optimization is in effect, and the participant rolls back its piece of work.

Coordinator Fails After Decision to Commit Transaction

If the coordinating thread completes the precommit phase, makes a decision to commit the transaction, but is terminated before the two-phase commit protocol can be completed, the coordinator-recovery mechanism goes into effect.

First, if a system failure occurs, the administrator must reinitialize shared memory and restart the database server; otherwise, coordinator recovery begins as part of the database server processing.

As part of its regular activity, the **main_loop** thread at the coordinator database server detects the following situations:

1. A two-phase commit protocol was under way.
2. The coordinating thread had reached a decision to commit. (Whether the coordinator sent messages to the participants to commit the transaction is ambiguous.)
3. The coordinating thread was terminated prematurely.

To complete the transaction, the **main_loop** thread forks a new coordinating thread. This new coordinating thread establishes a new connection at each participant database server.

The new coordinating thread sends a message to each participant thread to determine the status of the piece of work that was assigned to that participant database server. If the participant received a commit message from the coordinator before it failed, the participant would have committed the piece of work and removed the entry from its transaction table but would have no knowledge of the transaction. Thus, if the new coordinating thread receives a message that the transaction status is unknown, it assumes that the piece of work was already committed.

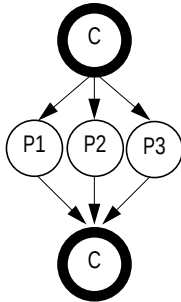
If the new coordinating thread receives a message that the transaction status is `can commit`, it sends a message to the participant to commit the transaction. After all participants send acknowledgments indicating that their piece of work is committed, the coordinating thread ends the transaction.

If the coordinating thread cannot contact one or more of the participants, or if execution errors are detected, messages are recorded in the database server message log.

The coordinating thread then continues its attempts to contact all participants and to complete the protocol. [Figure 30-5](#) illustrates the coordinator-recovery process.

Figure 30-5
Example of Automatic Coordinator Recovery

Start protocol



Coordinator:

Records decision to begin precommit phase. Sends a *prepare* message to the participants.

All Participants:

Determine that the commit can occur. Record the fact that a commit can occur. Return message: *can commit*.

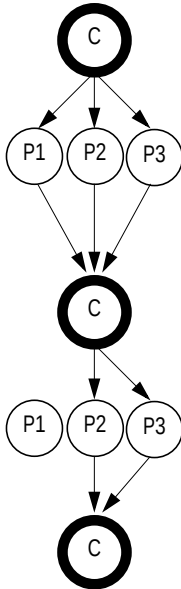
Coordinator:

Records decision to commit transaction.

Coordinator Fails

Start coordinator recovery

After the coordinator returns on-line, the `main_loop` thread forks a new coordinating thread.



Coordinator:

Queries each participant for transaction status.

Participant 1:

Reports *unknown status*.

Participants 2 and 3:

Determine that the commit can occur. Record the fact that the commit can occur. Return message *can commit*.

Coordinator:

Sends a *commit* message to participants 2 and 3. Participant 1 is assumed to have committed.

Participants 2 and 3:

Commit the piece of work. Record the commit. Remove transaction entry from shared-memory transaction table. Release shared-memory resources. Return confirmation message: *committed*.

Coordinator:

Records that the transaction is committed. Removes transaction entry from shared memory. Releases shared-memory resources.

End coordinator recovery

Automatic-Recovery Mechanisms for Participant Failure

Participant recovery occurs whenever a participant thread precommits a piece of work but 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.

The actual participant-recovery mechanism depends on whether the global transaction was committed or rolled back.

Participant Fails—Global Transaction Committed

In this scenario, the participant thread precommits its piece of work successfully and then unexpectedly terminates. The coordinator then decides to commit the global transaction. When the coordinator sends the terminated participant a message to commit, it receives an error indicating that the participant is down. At this point, the coordinator takes the following actions:

1. Drops the existing connection to the participant
2. Forks a new participant thread
3. Resends the message to commit the piece of work

If the participant is down, the coordinator is not able to fork a new participant thread. The coordinator continues to loop through this procedure, however, until it succeeds (when the system administrator brings the participant back on-line).

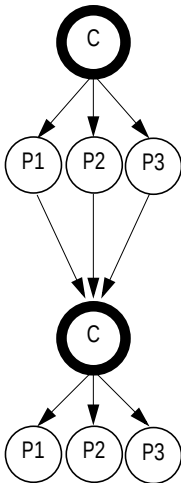
Participant Fails—Global Transaction Rolled Back

In this scenario, the participant thread precommits its piece of work successfully and then unexpectedly terminates. The coordinator then determines to roll back the global transaction. The coordinator sends a message to the terminated participant to roll back its piece of work. The participant does not receive the message because it is not running, but the coordinator in this scenario closes the global transaction anyway because a coordinator never waits for an acknowledgment on a rollback transaction.

When the system administrator brings the participant back up, the piece of work from the global transaction is still waiting for a message from the coordinator to commit or roll back. The participant thread, which is running the piece of work, waits for a period of time specified by the `TXTIMEOUT` variable and then forks a new participant thread. This new participant thread is responsible for determining the state of the global transaction. It forks a new thread on the coordinator and sends a message to this thread to determine the status of the global transaction. Because the coordinator previously made the decision to roll back the global transaction, no transaction status is in the coordinator shared memory. The participant thread rolls back its piece of work, therefore, and recovery is complete. [Figure 30-6 on page 30-16](#) illustrates this scenario.

Figure 30-6
Automatic Participant Recovery

Start protocol



Coordinator:

Records decision to begin precommit phase. Sends a *prepare* message to the participants.

Participant 1:

Returns message: *cannot commit*.

Participants 2 and 3:

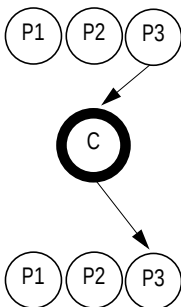
Return message: *can commit*.

Participant 3 Fails

Coordinator:

Receives *cannot commit* response. Records decision to roll back transaction. Rolls back work on behalf of transaction. Removes transaction entry from shared memory. Releases shared-memory resources. Sends a *rollback* message to each participant.

Start participant recovery



After Participant 3 returns on-line, the transaction on P3 waits for a message from the coordinator until TXTIMEOUT elapses. When TXTIMEOUT elapses, the participant forks a new coordinating thread.

Participant 3:

Queries coordinator about the transaction status.

Coordinator:

Reports that transaction status is unknown.

Participant 3:

Interprets status as instruction to roll back. Rolls back piece of work. Removes transaction entry from shared memory. Releases shared-memory resources.

End participant recovery

Race Conditions

A race condition might develop between the coordinator and the participant when the global transaction is committed.

For example, a race condition can result if a network failure occurs after the coordinator determines that the transaction should be committed. When the network comes back up, the coordinator tries to send the commit instruction to all the participants as described in [“Participant Fails—Global Transaction Committed” on page 30-14](#). If a participant does not receive a message from the coordinator before the TXTIMEOUT period, this participant attempts to contact the coordinator using the mechanism in [“Participant Fails—Global Transaction Rolled Back” on page 30-15](#). In this scenario, however, the global transaction is committed, so the participant finds a transaction status in the coordinator shared memory. In this case, the participant goes to sleep, allowing the coordinator time to send it the message to commit its piece of work.

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.

How Presumed-Abort Optimization Affects Automatic Recovery

Each participant database server must initiate automatic recovery if the coordinating thread fails in the following situations:

- Before it makes a decision to commit or roll back the transaction
- After it decides to roll back a transaction

This responsibility is part of the presumed-abort optimization. (See [“Automatic-Recovery Mechanisms for Coordinator Failure” on page 30-10.](#))

Why Optimization Is Realized

Optimization is realized because the coordinator is not required to flush the logical-log record (BEGPREP) that indicates a two-phase commit protocol has begun. If this information is lost (for example, if the coordinator fails before making a decision), each participant automatically rolls back its piece of work. Thus the logical log can be buffered, which represents the most significant part of the streamlined processing.

To a lesser extent, message traffic is reduced because the coordinator receives acknowledgment only when a transaction commits. Participants do not acknowledge rollbacks.

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 31, “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 Successful Completion of Transaction

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 on-line 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 30-22](#))
- Heuristic end transaction (described in [“The Heuristic End-Transaction Scenario” on page 30-26](#)).

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

Heuristic rollback is an independent action that either the database server or the administrator can take to roll 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-Water Mark

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-water mark (`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-water mark is reached (LTXEHW). 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 31-4.)

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 on-line 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-water mark.

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-water mark (LTXEHWM). If this point is reached, normal processing is suspended. At some point after the LTXEHWM high-water mark 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:

```
(timestamp) 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.

Tracking a Global Transaction

You can use the **onstat -x** utility to track transactions as they execute. For sample output of the **onstat -x** utility, refer to 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 30-23 . 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 30-23 . 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 30-20 .

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 31, “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*.

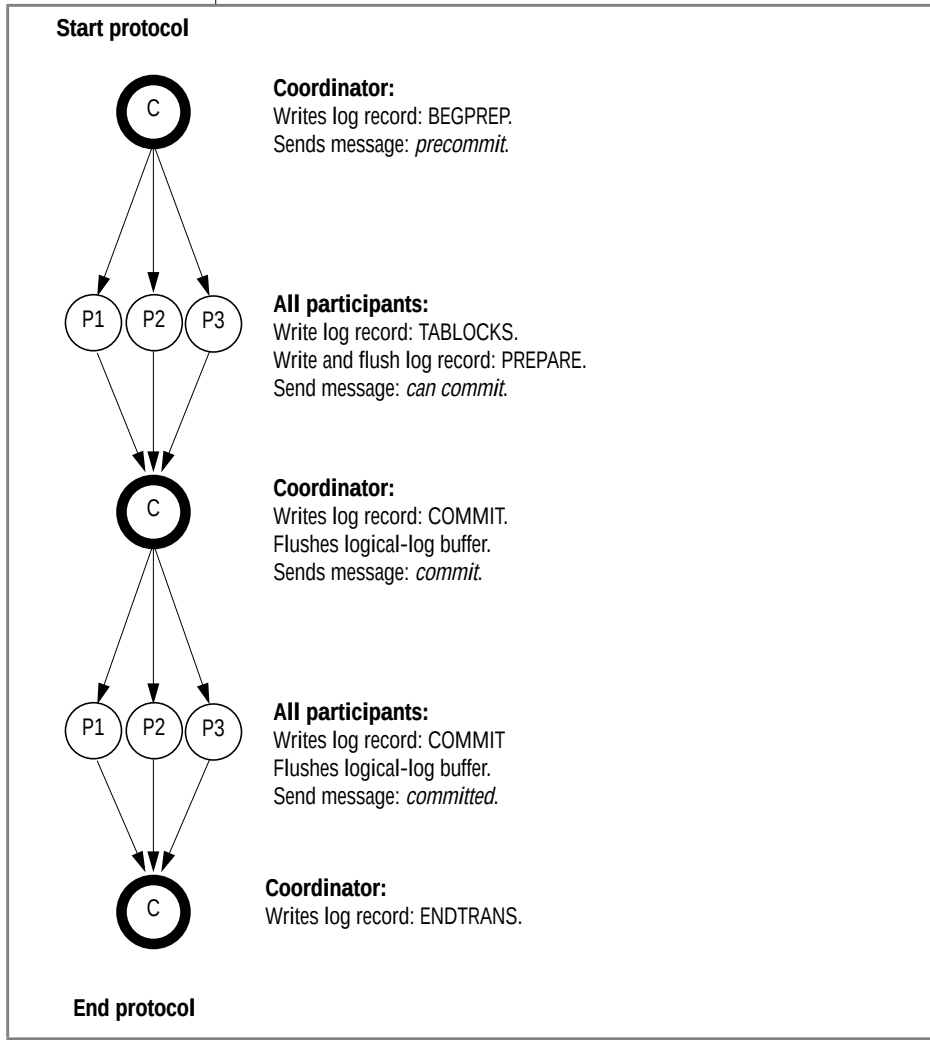
This section examines the sequence of logical-log records that are written during the following database server scenarios:

- 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 30-7 illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

Figure 30-7
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.

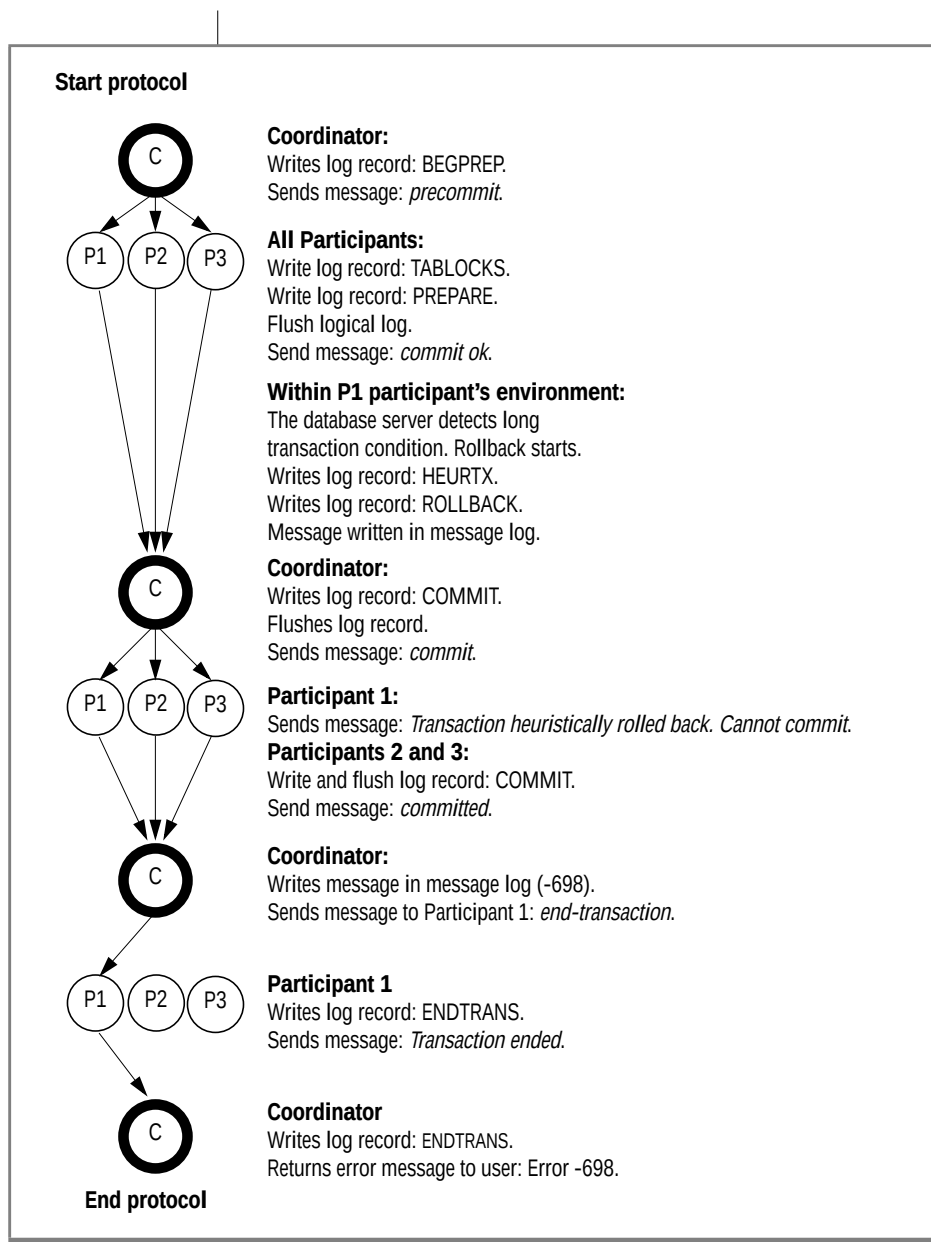
The participants in [Figure 30-7 on page 30-31](#) must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. 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 30-8 on page 30-34](#) 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 30-7 on page 30-31](#). 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 30-22](#).

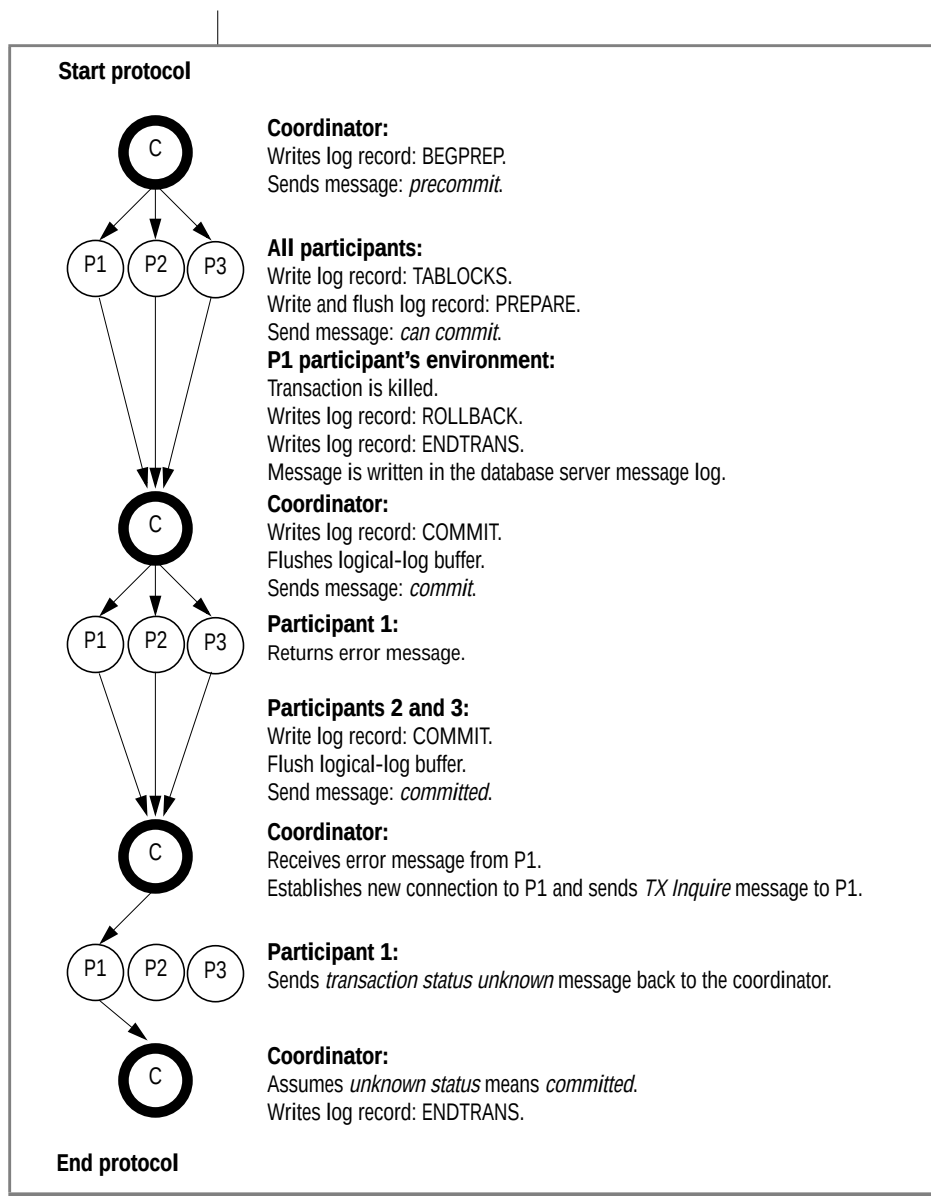
Figure 30-8
Logical-Log
Records Written
During a Heuristic
Rollback



Logical-Log Records Written After a Heuristic End Transaction

[Figure 30-9 on page 30-36](#) 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 30-9](#), 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 30-26](#).

Figure 30-9
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 30-10.](#)

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 Informix database 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 30-10 illustrates this scenario.

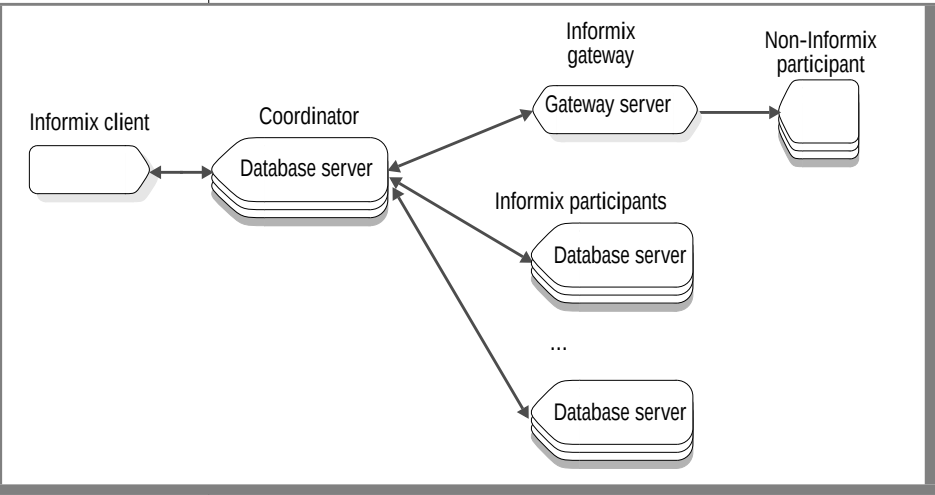


Figure 30-10
*Configuration That
Requires
Heterogeneous
Commit for
Distributed
Transactions*

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
Informix Enterprise Gateway with DRDA	IBM DB2, OS / 400, SQL / DS
INFORMIX-Enterprise Gateway for EDA / SQL	EDA / SQL
INFORMIX-Enterprise Gateway Manager	Any database server with ODBC connectivity

UNIX

Enabling and Disabling Heterogeneous Commit

Use one of the following methods to enable or disable heterogeneous commit:

- Use ON-Monitor. ♦
- Change the HETERO_COMMIT configuration parameter in your configuration file.

In either case, the change does not take effect until you reinitialize shared memory by bringing the database server off-line and then on-line again.

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, heterogeneous commit or two-phase commit, the transaction coordinator uses 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 30-7](#).

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 30-11 on page 30-42](#) illustrates this process.

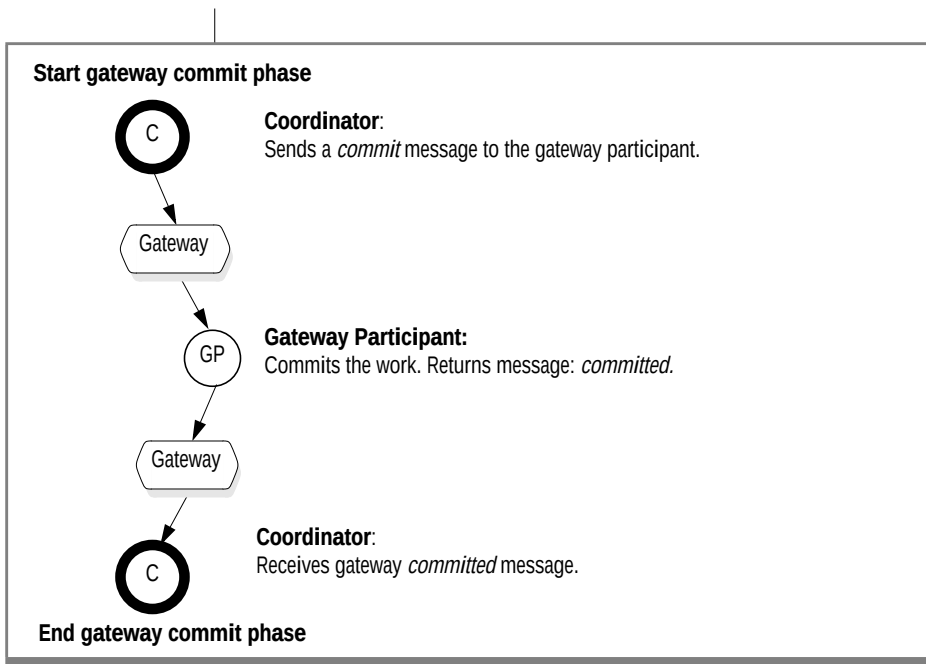


Figure 30-11
*Heterogeneous
Commit Phase That
Results in a
Committed
Transaction*

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as [Figure 30-11](#) 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 <code>commit transaction</code> message from coordinator, but before participant performs commit	Data consistency is maintained.
After participant receives <code>commit transaction</code> 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 30-44](#).

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

Determining If Manual Recovery Is Required.	31-3
Determining If a Transaction Was Implemented Inconsistently . . .	31-4
Global Transaction Killed Prematurely	31-4
Heuristic End Transaction.	31-4
Heuristic Rollback	31-5
Determining If the Distributed Database Contains Inconsistent Data	31-6
Obtaining Information from the Logical Log	31-7
Obtaining the Global Transaction Identifier.	31-8
Deciding If Action Is Needed to Correct the Situation	31-9
Example of Manual Recovery	31-10

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 30-18](#).) 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:

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 30-20.](#)) 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 30-26.](#)) 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 the 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 31-8](#).
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 31-1](#) 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 31-1
*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 31-2](#) with that of the BEGPREP record in [Figure 31-1](#).

```
c7064 184 PREPARE 4 0 c7038 chrisw
000000b8 00000044 00000004 000c7038 .....D .....p8
00005cd6 00000002 63687269 73770000 ..'.... chrisw..
00000000 00000000 00000069 00087eeb .....i...~.
00006b16 00000000 00000010 00ba5a10 ..k.....Z.
00000002 00ba3a0c 00000006 00000000 .....
00ba5a10 00ba5a1c 00000000 00000000 ..Z...Z. ....
00ba3a0e 00254554 00ba2090 00000001 ...:.%ET ..
00000000 00ab8148 0005fd70 00ab8148 .....H ...p...H
0005fe34 0000003c 00000000 00000000 ...4...< .....
00000000 00ab80cc 00000000 00ab80c4 .....
00ba002f 63687269 73770000 00120018 .../chrisw.....
00120018 00ba0000 .....
```

Figure 31-2
Output of the
onlog -I 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	len	type	xididlink						
.....									
17018	16	CKPOINT	0	0	13018	0			
18018	20	BEGIN	2	1	0	08/27/91 10:56:57	3482	nhowe	
1802c	32	HINSERT	2	0	18018	1000018	102	4	
1804c	40	CKPOINT	0	0	17018	1			
begin									
1	2	1	1802c						
19018	72	BEGPREP	2	0	1802c	6d69	1		
19060	16	COMMIT	2	0	19018	08/27/91 11:01:38			
1a018	16	ENDTRANS	2	0	19060	580543			

The following excerpt is taken from the logical log at the database server **apex**:

addr	len	type	xid	id	link				
....									
16018	20	BEGIN	2	10		08/27/91	10:57:07	3483	pault
1602c	32	HINSERT	2		0	16018	1000018	102	4
1604c	68	PREPARE	2		0	1602c	eh		
17018	16	HEURTX	2		0	1604c		1	
17028	12	CLR	2		0	1602c			
17034	16	ROLLBACK	2		0	17018	08/27/91	11:01:22	
17044	40	CKPOINT	0		0	15018		1	
		begin	xid		id	addr		user	
	1		2		1	17034	-----		
18018	16	ENDTRANS	2		0	17034		8806c3	
....									

First, you would try to match the transactions in the current database server log with the transactions in the **apex** database server log. The BEGPREP and PREPARE log records each contain the GTRID. You can extract the GTRID by using **onlog -l** and looking at the data portion of the BEGPREP and PREPARE log records. The GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple, though less precise, approach is to look at the time of the COMMIT or ROLLBACK records. The times 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 **sysables** 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
```

Index

Numerics

- 64-bit addressing
 - and buffer pool 12-20
 - description 12-59

A

- Access method
 - definition 1-25
- Adding listen threads 10-37
- Adding virtual processors 10-17
- ADM (administration) virtual processor 10-17
- Administrative tasks
 - configuration tasks 2-4
 - consistency checking 29-3
 - controlling location of
 - storage 14-19
 - cron jobs 3-21
 - initial tasks 2-4
 - list of tasks 3-18
 - of database administrator 1-4
 - of database server
 - administrator 1-4
 - of database server operator 1-4
 - planning 3-4
 - routine tasks 2-7
 - startup and shutdown
 - scripts 3-18
 - types of 2-3
- ADT (audit) virtual processor 10-39
- ADTERR parameter 4-19
- ADTMODE parameter 4-19, 10-39
- ADTPATH parameter 4-19
- ADTSIZE parameter 4-19
- AFF_NPROCS parameter 4-15
 - description of 4-15
 - purpose of 10-23
- AFF_SPROC parameter 4-15, 10-23
- Aggregate, parallel processing
 - of 10-10
- AIO virtual processors
 - how many 10-31
 - NUMAIOVPS parameter 10-31
- ALARMPROGRAM
 - parameter 2-14
- Aliases. *See* DBSERVERALIASES parameter.
- ALL keyword, with
 - DATASKIP 15-42
- Allocating
 - cooked file space 15-9
 - initial chunk 15-10
 - metadata 15-24
 - sbspaces 14-34
- Allocating disk space
 - extent 14-17
 - for mirrored data 25-5
 - procedure 15-6
- Allocating disks 3-7
- ALTER FRAGMENT statement
 - ATTACH clause 16-5
 - Attaching fragments 16-10
 - dropping a fragment 12-21
- ALTER TABLE statement
 - and smart large objects 16-7
 - mentioned 5-47
- ANSI compliance
 - level Intro-15
- ANSI-compliant transaction
 - logging. *See* Logging.

Application developer, role of 1-4
 Applications, types of 16-16
 Application, client. *See* Client application.
 archecker utility
 description 1-14
 See also Backup and Restore Guide.
 Archive
 preparing environment in multiple-residency setting 7-11
 with multiple residency 7-11
 ASF. *See* Connectivity.
 Assertion failure
 and data corruption 29-9
 description of 29-3
 determining cause of 29-11
 during consistency checking 29-6
 during processing of user request 29-8
 form in message log 29-6
 Assertion failure file
 af.xxx 29-7
 gcore.xxx 29-7
 list of 29-7
 shmem.xxx 29-7
 Asterisk, as wildcard in hostname field 5-43
 Asynchronous I/O
 description of 10-29
 kernel (KAIO) 10-27
 write requests for mirrored data 24-9
 Attaching fragments 16-5
 Attaching to shared memory
 database server utilities 12-12
 description 12-11
 virtual processors 12-12
 virtual processors and key value 12-13
 Audit mode 10-39
 Auditing 1-17
 configuration parameters 4-19
 Authentication
 default policy 5-14
 description of 5-13
 Automatic backups
 and ISM 26-6

Automatic recovery, by two-phase commit protocol 30-10
 Availability
 and critical data 14-19
 as goal in efficient disk layout 14-48
 improving with fragmentation 16-5
 sample disk layout 14-56
 AVG_LO_SIZE tag, onspaces 14-30

B

Backing up
 blobspaces 15-21
 changing database logging 15-13
 chunks 15-18
 dbspaces 15-15
 log files 15-13, 20-4
 physical log 15-13
 sbspaces 15-25
 Backup
 active users, and 1-14
 and checkpoints 23-11
 and fragmentation 16-5
 automatic 26-6
 dbspace backup, mentioned 1-14
 description of 2-8
 external 1-15, 2-11
 freeing a log file 20-13
 mentioned 14-47
 of TEXT and BYTE data 14-24
 reducing size of 14-23
 sbspaces 14-34
 strategy 3-4
 transaction records 1-14
 verification with archecker 1-14
 verifying 1-14
 with multiple residency 7-11
 Bad-sector mapping, absence of 1-16, 15-63
 base 14-36
 obBefore-image
 contents 23-17, 23-21
 described 21-3
 flushing of 12-46
 in physical log buffer 12-46
 journal. *See* Physical log.
 role in buffer modification 12-45
 Big buffers, description of 12-30
 Binary large object. *See* BLOB data type.
 Binding CPU virtual processors
 benefit of 10-12
 parameters 10-23
 Bit-map page
 component of a index tblspace 14-42
 component of a tblspace 14-42
 BLOB data type
 mentioned 1-22, 14-24
 See also Smart large object.
 Blobpage
 description 14-12
 determining fullness 15-35
 freeing deleted pages 19-26
 fullness terminology explained 15-36
 interpreting average fullness 15-36
 oncheck -pB display explained 15-35
 performance considerations 15-33
 relationship to chunk 14-13
 size and storage efficiency 15-34
 sizing recommendations 15-21
 storage statistics 15-34
 writes, bypassing shared memory 12-55
 Blobspace
 activating 19-26
 adding a chunk 15-17
 backing up 15-13, 15-21
 blobpage time stamps 23-13
 buffers 12-56
 creating with ON-Monitor 15-23
 creating with onspaces 15-23
 description 1-22, 14-24
 dropping with ON-Monitor 15-33
 dropping with onspaces 15-33
 dropping, initial tasks 15-31
 free-map page
 role in blobpage logging 19-28
 role in tracking blobpages 12-57

- logging 14-24
- logical-log administration tasks
 - for 19-26
- multiple residency 7-10
- names 15-15
- overhead pages 19-27
- performance
 - considerations 15-34
 - storage statistics 15-34
 - writing data to 12-55
 - writing TEXT and BYTE data 12-57
- Blobspace blobpage
 - and physical logging 21-5
- Block device 14-7
- Boldface type Intro-8
- Boot file. *See* Startup script.
- B-tree
 - cleaner thread 12-30
- B-tree index. *See* Performance Guide.
- Buffer
 - big buffers 12-30
 - blobpage buffer 12-56
 - concurrent access 12-42
 - current lock-access level for 12-27
 - data replication 12-23, 27-10
 - dirty 12-45
 - exclusive mode 12-36
 - flushing 12-45
 - how a thread accesses a buffer page 12-42
 - how a user thread acquires 12-38
 - how thread releases after modification 12-45
 - least-recently used 12-37
 - lock types 12-35
 - lock-access level of 12-43
 - logical-log buffer 12-21, 12-45
 - monitoring statistics and use of 13-21
 - most-recently used 12-37
 - not dirty 12-45
 - physical-log buffer 12-23, 12-45
 - reading from disk 12-43
 - releasing if no thread waiting 12-44
 - releasing if thread waiting 12-44
 - releasing when modified 12-44

- releasing when not modified 12-44
- share lock 12-36
- smart large objects 12-58, 14-32
- status 12-27
- synchronizing flushing 12-46
- threads waiting for 12-27
- what occurs when modified 12-44
- write types during flushing 12-50
- Buffer flushing
 - description of 12-45
 - how synchronized 12-49
- Buffer pool
 - 64-bit addressing 12-20
 - bypassed by blobspace data 12-55
 - contents of 12-18
 - description of 12-18
 - flushing 12-46
 - full checkpoint 23-5
 - fuzzy checkpoint 23-5, 23-6
 - LRU queues management 12-37
 - minimum requirement 12-19
 - monitoring activity 13-25
 - read-ahead 12-41
 - size of buffer 12-20
 - smart large objects 12-58
 - synchronizing buffer flushing 12-49
- Buffer table
 - contents of 12-27
 - description of 12-27
 - LRU queues 12-36
- Buffered logging, when flushed 17-13
- Buffered transaction logging. *See* Logging.
- BUFFERING tag, onspaces 14-32
- Buffers
 - smart large objects and 14-32
- BUFFERS parameter
 - 64-bit addressing 12-20
 - description 12-19
 - smart large objects and 12-58
- Buffer-size option, in options field 5-35
- BUFSIZE. *See* Page size.
- Built-in data types
 - description of 1-20

- fuzzy operations 23-5
- mentioned 1-5
- BYTE data type
 - Committed Read isolation 23-14
 - mentioned 1-22
 - See also* TEXT and BYTE data; Simple large object.
- Byte-range locking 12-24, 14-33

C

- C user-defined routine
 - mentioned 1-24
- Cache
 - Optical Subsystem memory 15-58
 - shared-memory buffer monitoring 13-21, 13-24
 - SPL routine cache
 - hash size 4-11, 13-13
 - SPL routine cache size 4-11, 13-13
 - SQL statement cache 4-12, 13-14
 - size 4-12, 13-14
- Cache. *See* Shared-memory buffer pool.
- Calculating size
 - blobpages 15-21
 - metadata 14-34, 15-25
 - page size 15-22
 - root dbspace 14-45
 - smart large objects 14-30
- Cascading deletes 17-5
- Casting, user-defined, mentioned 1-20
- Central registry, sqlhosts 5-24
- Character 2
- Character large object. *See* CLOB data type.
- Character-special devices 1-11
- Checkpoint
 - and chunk writes 12-52
 - and flushing of regular buffers 12-46
 - and logical-log buffer 12-22
 - and logical-log file 23-7
 - and physical-log buffer 12-47, 21-11, 23-7
 - backup considerations 23-11
 - description of 23-4

- events that initiate 23-7
- full 19-31, 23-5
- fuzzy 23-5
- how it works 23-9 to 23-11
- mentioned 9-9
- monitoring activity 23-24
- role in data replication 27-15
- role in fast recovery 23-17, 23-18
- step in shared-memory initialization 9-9
- Chunk
 - activity during mirror recovery 24-9
 - adding to a blob space 15-17
 - adding to a db space 15-17
 - adding to a sb space 15-27
 - adding to an sb space 14-34
 - adding to mirrored db space 25-11
 - adding with ON-Monitor 15-19
 - adding with onspaces 15-19
 - allocating initial 15-10
 - and associated partitions 14-49
 - backing up 15-18
 - changing mirror chunk status 25-8
 - checking status 15-46, 15-47, 29-9
 - concepts 14-5
 - creating a link to the pathname 3-8, 15-11, 15-12, 25-6
 - dropping from a blob space 15-30
 - dropping from a db space 15-30
 - dropping from an sb space 15-30
 - exceeding size limits with LVM 14-61
 - general disk layout guidelines 14-49
 - I/O errors during processing 29-9
 - limits on size and number 14-6
 - maximum size 15-19
 - monitoring 15-46, 15-48, 29-9
 - multiple residency 7-10
 - name, when allocated as raw device 14-7
 - recovering a down chunk 25-8
 - relation to extent 14-17
 - specifying metadata area 15-27
 - Chunk free list
 - combining and splitting pages 17-8
 - monitoring 22-4
 - Chunk table
 - and mirroring 12-27
 - contents of 12-28
 - description of 12-27
 - Chunk write
 - checkpoints 12-52
 - monitoring 13-26
 - CKPTINTVL configuration parameter 4-11
 - CKPTINTVL parameter
 - description of 4-11
 - initiating checkpoints 23-7
 - Classes of virtual processor 10-6
 - CLEANERS configuration parameter 4-12
 - CLEANERS parameter
 - description of 4-12
 - purpose of 12-28
 - Client
 - and MSCS 26-5
 - connecting to primary server 27-5
 - connecting to secondary server 27-5
 - description of 1-5
 - designing for redirection 27-21
 - enabling communication 5-14
 - finding and connecting to a host 5-23
 - improving performance and availability 27-4
 - loopback connection 5-13
 - reacting to failure 27-18
 - redirecting and connectivity in data replication 27-20
 - redirecting in data replication 27-19
 - remote 5-18
 - security requirements 5-18
 - specifying a dbservername 5-47
 - Client application
 - redirecting in data replication 27-20
 - testing 6-4
 - wildcard addressing 5-45
 - Client/server architecture, description of 1-5
 - Client/server configuration example
 - local loopback 5-51
 - multiple connection types 5-53
 - multiple database servers 5-55
 - multiple residency 5-55
 - network connection 5-52
 - shared memory 5-50
 - using IPX/SPX 5-53
 - listen and poll threads 10-34
 - local loopback 5-13
 - shared memory 5-11
 - CLOB data type
 - mentioned 1-22
 - See also* Smart large object.
 - Code set, ISO 8859-1 Intro-4
 - Code, sample, conventions for Intro-11
 - Comment icons Intro-10
 - COMMIT statement, unbuffered logging 17-12
 - Committed Read isolation level
 - data-consistency checks 23-14
 - role of blob time stamps 23-13
 - Commit, heterogeneous 30-38
 - Communication configuration file.
 - See* ONCONFIG configuration file.
 - Communication support module
 - password encryption 5-14, 5-22
 - sqlhosts option field 5-36
 - Communication Support Modules 5-14
 - Communication support services
 - description of 5-13
 - message confidentiality 5-13
 - message integrity 5-13
 - Communications portion of shared memory
 - contents of 12-33
 - description of 12-33
 - how client attaches 12-11
 - size of 12-34
 - Communication, client to database server. *See* Connectivity.
 - Complex data types 1-20, 1-21

- Compliance
 - with industry standards Intro-15
- conscm.cfg file
 - configuring communication
 - support module 5-14
 - entry for password
 - encryption 5-22
 - format of entries 5-21
 - location of 5-21
- Concurrency
 - improving with
 - fragmentation 16-5
- Concurrency control 12-34
- Confidentiality, of communication
 - messages 5-13
- Configuration
 - dbspaces 15-13
 - estimating required disk
 - space 14-48
 - monitoring 3-23
 - planning for the database
 - server 3-4
 - storage devices 3-22
- Configuration file
 - and multiple residency 6-3, 7-8
 - connectivity 5-14
 - onconfig.std 3-14
 - warning about not modifying
 - onconfig.std 3-15
- Configuration parameter
 - ADTERR 4-19
 - ADTMODE 4-19
 - ADTPATH 4-19
 - ADTSIZE 4-19
 - AFF_NPROCS 4-15
 - AFF_SPROC 4-15
 - ALARMPROGRAM 2-14
 - BUFFERS 12-19
 - CKPTINTVL 4-11, 23-7
 - CLEANERS 4-12
 - DATASKIP 15-39
 - DBSERVERALIASES 5-47
 - DBSERVERNAME 5-28, 5-47
 - DBSPACETEMP 14-50, 15-15
 - DRINTERVAL 4-17
 - DRLOSTFOUND 4-17
 - DRTIMEOUT 4-17
 - DUMPCNT 29-11
 - DUMPCORE 29-11
 - DUMPDIR 29-7, 29-11
 - DUMPGCORE 29-7, 29-11
 - DUMPSHMEM 29-7, 29-11
 - IDX_RA_PAGES 12-41
 - LOGBUFF 12-22
 - LRUS 12-38
 - LRU_MAX_DIRTY 12-39
 - LTXHWM 12-53
 - MIRROROFFSET 15-8
 - NETTYPE 5-48
 - OFF_RECOVERY_THREADS
 - 4-16
 - ON_RECOVERY_THREADS
 - 4-16
 - OPCACHEMAX 15-58
 - OPTCOMPIND 16-22
 - PC_HASHSIZE 4-11, 13-13
 - PC_POOLSIZE 4-11, 13-13
 - RA_THRESHOLD 12-41
 - RESIDENT 13-17
 - ROOTOFFSET 15-8
 - SBSPACENAME 14-27
 - SERVERNUM 12-12, 12-13
 - SHMADD 4-10, 12-25
 - SHMBASE 4-10, 12-12, 12-14, 13-6
 - SHMTOTAL 4-10, 12-9
 - SHMVIRTSIZE 4-10, 12-25
 - STACKSIZE 12-31
 - STMT_CACHE 4-12, 13-14
 - STMT_CACHE_SIZE 4-12, 13-14

See also Administrator's Reference;
 Configuration parameter use;
 and individual parameter names.

- Configuration parameter use
 - and initial chunk of root
 - dbspace 14-22
 - enabling Logs Full HWM 19-22
 - for diagnostic information 29-11
 - for multiple residency 6-5
 - shared memory 13-3
- Configuring multiple ports 5-17
- CONNECT statement
 - example 5-48
 - mentioned 5-47
- Connecting
 - and multiple residency 7-10
 - description of 10-34
 - methods 10-32
 - to multiple database servers 5-55
 - to multiple servers 6-3
 - to non-Informix databases 1-18
- Connection
 - database versus network 5-7
 - IPX/SPX 5-53
 - local loopback, definition of 5-13
 - local loopback, example 5-51
 - multiple connection types
 - example 5-53
 - multiple residency, example 5-55
 - multiplexed 5-7
 - network, example 5-52
 - security restrictions 5-18
 - shared memory, description
 - of 5-11
 - TCP/IP 5-18
- Connectivity
 - ASF 5-6
 - configuration file 5-14
 - configuration parameters 5-46
 - hosts file 5-16
 - services file 5-16
- Connectivity file
 - sqlhosts 3-14
- Consistency checking
 - corruption of data 29-8
 - data and overhead 29-4
 - index corruption 29-9
 - monitoring for data
 - inconsistency 29-6
 - overview 29-3
 - periodic tasks 29-3
 - sbspaces 14-35
 - validating indexes 29-5
- Console messages 2-15
- CONSOLE parameter
 - changing 2-15
- Constraint, deferred checking 17-5
- Contact information Intro-15
- Contention. *See* Disk contention.
- Context switching
 - description of 10-14
 - how functions when OS
 - controls 10-10
 - OS versus multithreaded 10-10
- Contiguous space for physical
 - log 22-4
- Control structures
 - description of 10-13

queues 10-16
 session control block 10-13
 stacks 10-15
 thread control block 10-13
 Conventions,
 documentation Intro-8
 Conversion
 during initialization 9-9
 Cooked file space
 allocating 15-9
 and buffering 14-9
 compared with raw space 14-7,
 14-8
 contiguity of space 14-9
 database server management
 of 14-8
 description of 14-7
 for static data 14-9
 rationale for using 14-9
 reliability 14-9
 warning 14-8
 Cooked files 1-12
 Coordinating database server
 and automatic recovery 30-11
 description of 30-5
 Coordinator recovery
 mechanism 30-11
 Core dump
 contents of gcore.xxx 29-7
 when useful 29-11
 See also DUMPCNT; DUMPDIR;
 DUMPGCORE;
 DUMPSHMEM.
 Corruption
 corrective actions 29-9
 determining if exists 29-9
 I/O errors from a chunk 29-9
 symptoms of 29-9
 CPU
 binding to virtual processor 10-12
 relationship to virtual
 processor 10-4
 CPU virtual processor
 adding and dropping in on-line
 mode 10-22, 10-27
 AFF_NPROCS parameter 10-23
 AFF_SPROC parameter 10-23
 and poll threads 10-32, 10-33
 DataBlade module on 10-21

 description of 10-19
 how many 10-20
 NUMCPUVPS parameter 10-20
 on a multiprocessor
 computer 10-21
 on a single-processor
 computer 10-21, 10-22
 preventing priority aging 10-22
 restrictions on 10-25
 the NUMCPUVPS
 parameter 10-20
 types of threads run by 10-19
 user-defined routine on 10-21
 CREATE DATABASE statement
 IN dbspace option 14-20
 CREATE FUNCTION
 statement 10-26
 CREATE TABLE statement
 and smart large objects 16-7
 FRAGMENT BY clause 16-10
 IN dbspace option 14-20
 mentioned 5-47
 specifying sbspaces in a PUT
 clause 14-27
 TEMP 16-9
 Creating 15-37
 blobspaces 15-23
 dbspaces 15-10
 fragmented tables 16-10
 sbspaces 15-24
 smart large objects 14-27
 tables with CLOB or BLOB data
 types 14-34
 temporary dbspaces 15-16
 Critical dbspaces
 mirroring 14-49
 storage of 14-19
 Critical media 1-16
 mirroring 14-59
 Critical section of code
 and checkpoints 23-10
 and filling of physical-log
 buffer 12-48
 description of 23-4
 cron jobs, warning about 3-21
 CSM 5-14

D

Data block. *See* Page.
 Data consistency
 fast recovery 23-14
 how achieved 23-3
 monitoring for 29-6
 symptoms of corruption 29-9
 time stamps 23-12
 verifying consistency 29-4
 Data definition statements, when
 logged 17-7
 Data files. *See* Logging.
 Data management 14-8
 Data manipulation statements,
 when logged 17-8
 Data mart 1-7
 Data replication
 actions to take if primary
 fails 27-19
 actions to take if secondary
 fails 27-19
 administration of 28-16
 advantages of 27-4
 changing database server
 mode 28-21
 changing database server
 type 28-22
 client redirection
 comparison of different
 methods 27-30
 handling within an
 application 27-28
 using DBPATH 27-21
 using
 INFORMIXSERVER 27-27
 using sqlhosts file 27-23
 configuring connectivity for 28-9
 database and data requirements
 for 28-6
 database server configuration
 requirements for 28-6
 description of 1-16, 27-4
 designing clients to use primary
 database server 27-31
 designing clients to use secondary
 database server 27-32
 detecting failures of 27-17
 DRINTERVAL parameter 27-11

DRLOSTFOUND
 parameter 27-12
 DRTIMEOUT parameter 27-17
 Enterprise Replication 2-6
 flush interval 4-17
 hardware and operating-system
 requirements for 28-5
 HDR overview 2-6
 how it works 27-8
 how updates are replicated 27-9
 initial replication 27-8
 lost-and-found transactions 27-12
 manual switchover 27-19
 mentioned 17-5
 monitoring status 28-24
 planning for 28-4
 possible failures 27-16
 read-only mode 8-4
 restarting after failure 27-20,
 28-29
 restarting after network
 failure 28-33
 restoring system after media
 failure 28-26
 role of checkpoint 27-15
 role of log records 27-9
 role of primary database
 server 27-5
 role of secondary database
 server 27-5
 role of temporary dbspaces 27-35
 setting up 28-5
 specialized threads 27-14
 starting 28-11
 synchronization 27-15
 wait time for response 4-17
 with asynchronous
 updating 27-11
 with synchronous updating 27-11
 Data storage
 concepts 14-3
 control of 14-20, 14-27
 limits on due to maximum chunk
 size 14-10
 maximum chunk size and
 number 15-19
 types of 14-6
 See also Disk space.

Data Type segment
 See also Disk space.
 Data types
 BLOB, mentioned 1-22
 built-in, description of 1-20
 built-in, mentioned 1-5
 BYTE, mentioned 1-22
 CLOB and BLOB 14-27, 23-6
 CLOB, mentioned 1-22
 complex, description of 1-21
 complex, mentioned 1-20
 distinct 1-21
 fuzzy operations 23-5
 nonfuzzy operations 23-6
 opaque 1-21
 opaque, mentioned 1-21
 row 1-22
 TEXT, mentioned 1-22
 user-defined 14-27
 user-defined, mentioned 1-18,
 1-20, 1-21
 Data warehouse 1-7
 Database
 description of 14-36
 estimating size of 14-48
 fragmentation 14-37
 location of 14-36
 migration. *See Informix Migration
 Guide.*
 monitoring 15-43, 18-11
 purpose of 14-36
 recovery. *See* Recovery.
 size limits 14-37
 tuning. *See* Performance tuning.
 Database administrator
 role of 1-4
 See also Administrative tasks.
 Database I/O 10-30
 Database logging
 and backups 15-13
 Database logging status
 ANSI-compliant,
 description 17-13
 buffered, description 17-13
 canceling logging
 using ON-Archive 18-7
 using ondblog 18-6
 changes permitted 18-4
 changes, general info 18-5

 changing buffering status
 using ON-Archive 18-8
 using ondblog 18-5
 using ontape 18-10
 description of 17-12
 distributed queries 17-14
 ending logging
 using ON-Archive 18-8
 using ondblog 18-6
 using ontape 18-9
 making ANSI-compliant
 using ON-Archive 18-8
 using ondblog 18-6
 using ontape 18-10
 modifying
 using ON-Archive 18-6
 using ondblog 18-5
 using ON-Monitor 18-11
 using ontape 18-9
 setting 17-6
 turning on logging
 using ON-Archive 18-7
 using ontape 18-9
 unbuffered, description 17-12
 who can change 17-15
 Database schema. *See Informix
 Migration Guide.*
 Database server
 client/server architecture 1-5
 connecting to multiple 5-55, 6-3
 distributed queries 1-18
 external spaces, description
 of 1-23
 fault-tolerant features 1-13
 high performance of 1-11
 installation script 3-10
 multimedia support 1-22
 multiple instances 6-3
 multithreaded 1-10, 10-4
 private installation 3-10
 remote 5-19
 scalability of 1-10
 security 1-17
 upgrading 3-10
 user-defined routines, description
 of 1-24
 Database server administrator 1-4
 Database server name. *See*
 dbservername.

- DATABASE statement,
 - mentioned 5-47
- DataBlade API
 - accessing smart large objects 14-25, 14-34
 - smart large object size 14-14
- DataBlade developer, role of 1-4
- DataBlade modules
 - description of 1-24
 - mentioned 1-18
 - virtual processors for 10-21
- Data-recovery mechanisms
 - fast recovery 23-14
 - smart large objects 14-25
- Data-replication buffer 12-23, 27-10
- DATASKIP parameter 15-39
 - using 15-39
 - with ALL keyword 15-42
- Data, estimating disk space
 - for 14-47
- DATE and DATETIME values 1-17
- DB-Access utility Intro-5
- DBCENTURY environment
 - variable 1-17
- DBEXPORT. *See Informix Migration Guide.*
- DBIMPORT. *See Informix Migration Guide.*
- DBPATH environment variable
 - and dbserver group 5-39
 - use in automatic redirection 27-21
- DBSCHEMA. *See Informix Migration Guide.*
- Dbserver group
 - purpose of 5-37
- DBSERVERALIASES parameter
 - and multiple residency 6-5
 - description of 5-47
 - example 5-47
 - in sqlhosts file 5-28
 - multiple connection types example 5-53
- DBSERVERNAME parameter
 - and multiple residency 6-5
 - associated protocol 10-33
 - database server ID 4-6
 - description of 5-47
 - in sqlhosts file 5-28
- multiple residency 7-8
- syntax rules 5-28
- virtual processor for poll thread 10-33
- Dbospace
 - adding a chunk 15-17, 15-19
 - adding a mirrored chunk 25-11
 - as link between logical and physical units of storage 14-19
 - backing up 15-13, 15-15
 - blobpage time stamps 23-13
 - configuration 15-13
 - creating 15-10
 - creating a temporary 15-16
 - creating during initial configuration 3-22
 - creating with ON-Monitor 15-17
 - creating with onspaces 15-16
 - description of 14-19
 - dropping a chunk from 15-29
 - dropping with ON-Monitor 15-33
 - dropping with onspaces 15-33
 - dropping, overview 15-31
 - initial dbospace 14-22
 - mirroring if logical-log files included 24-8
 - monitoring simple large objects 15-61
 - multiple residency 7-10
 - names 15-15
 - purpose of 14-19
 - role in fragmentation 16-3
 - root dbospace defined 14-22
 - shared-memory table 12-28
 - smart large objects 14-15
 - temporary 14-22
- Dbospace table
 - contents of 12-28
 - description of 12-28
 - metadata 12-28
- DBSPACETEMP environment
 - variable 14-41, 15-15
- DBSPACETEMP parameter 15-15
 - and load balancing 14-50
 - if not set 14-41
 - relationship to DBSPACETEMP environment variable 14-40
- Deadlock prevention 19-24
- DEADLOCK_TIMEOUT parameter
 - in two-phase commit 30-37
- Decision-support application
 - characteristics 16-19, 16-20
 - definition of 16-18
 - uses of 1-7, 16-18
- Decision-support query
 - DS_MAX_QUERIES parameter 4-13
 - See also* PDQ.
- Default configuration file 3-15, 9-7
- DEFAULT keyword, with SET DATASKIP 15-40
- Default locale Intro-4
- Deferred checking of constraints 17-5
- Deleting
 - smart-large-object data 15-31, 15-33
- Demonstration databases Intro-5
- Dependencies, software Intro-4
- Detached index 16-9
- Device 14-7
 - when offsets are needed 15-7
- Diagnostic information
 - and disk space restraints 29-11
 - collecting 29-11
 - parameters to set 29-11
- Diagnostic messages. *See* Message log.
- Dictionary cache 12-32
- Dirty buffer, description of 12-45
- Dirty Read isolation level
 - and locking 27-34
 - data-consistency checks 23-14
 - role of blob time stamps 23-13
- Disabling I/O error
 - circumstances under which they occur 14-18
 - defined 14-18
 - destructive versus nondestructive 14-18
 - monitoring with event alarms 15-45
 - monitoring with message log 15-44
- Disk allocation 3-7
- Disk configuration 3-5

Disk contention
 and high-use tables 14-52
 of critical media 14-50
 reducing 14-48

Disk failure 1-14

Disk I/O
 errors during processing 29-9
 kernel asynchronous I/O 10-27
 logical log 10-27
 operating system I/O 14-7
 physical log 10-27
 priorities 10-28
 queues 10-31
 raw I/O 14-7
 reads from mirrored
 chunks 24-10
 role of shared memory in
 reducing 12-5
 smart large objects 14-25
 virtualprocessor classes 10-27
 writes to mirrored chunks 24-9

Disk layout
 and archiving 14-51, 14-54
 and logical volume
 managers 14-61
 and mirroring 14-54
 and table isolation 14-52
 for optimum performance 14-49,
 14-50
 sample disk layouts 14-56
 trade-offs 14-56

Disk management 3-7

Disk page
 before-images in physical
 log 12-46
 function of time-stamp
 pairs 23-12
 read ahead 12-41

Disk space
 allocating cooked file space 15-9
 allocating raw disk space
 on UNIX 14-9
 on Windows NT 15-12
 caution with multiple
 residency 7-10
 configuring for multiple
 residency 6-6
 creating a link to chunk
 pathname 3-8, 15-11, 15-12

description of 15-6
 estimating size of 14-45
 initialization 9-8
 definition of 9-3, 15-6
 with new database server 3-17
 layout guidelines 14-48
 limits on due to maximum chunk
 size 3-8
 maximum chunk size 15-19
 multiple residency 7-10
 offsets for chunk pathnames 15-7
 optimizing temporary space
 layout 14-50
 performance
 considerations 14-48, 14-55
 requirements 14-48
 storage of TEXT and BYTE
 data 15-35
 temporary. *See* Temporary disk
 space.
 tracking usage by tblspace 14-42

Distinct type, description of 1-21

Distributed databases 1-18

Distributed queries, description
 of 1-18

Distributed transaction
 and two-phase commit
 protocol 30-3
 determining if inconsistently
 implemented 31-6
 mentioned 17-5

Distribution scheme 16-6

Documentation notes Intro-14

Documentation, types of
 documentation notes Intro-14
 error message files Intro-13
 machine notes Intro-14
 on-line help Intro-13
 on-line manuals Intro-12
 printed manuals Intro-13
 related reading Intro-15
 release notes Intro-14

Domain, Windows NT
 and database server 5-5
 and user accounts 5-5
 controller 5-5
 description of 5-5
 trusted 5-6
 user account in 5-6

DRINTERVAL parameter
 description of 4-17
 setting for asynchronous
 updating 27-11
 setting for synchronous
 updating 27-11

DRLOSTFOUND parameter
 description of 4-17
 use with data replication 27-12

Dropping
 extspaces 15-38
 sbspaces 15-31

Dropping fragments 12-21

DRTIMEOUT parameter
 description of 4-17
 role in detecting data replication
 failures 27-17

DSA. *See* Dynamic Scalable
 Architecture.

DSS application. *See* Decision-
 support application.

DS_MAX_QUERIES parameter
 description of 4-13

DS_MAX_SCANS parameter
 description of 4-13

DS_TOTAL_MEMORY parameter
 description of 4-13

DUMPCNT parameter 29-11

DUMPCORE parameter 29-11

DUMPDIR parameter 29-7, 29-11

DUMPGCORE parameter 29-7,
 29-11

DUMPSHMEM parameter 29-7,
 29-11

Dynamic Host Configuration
 Product 5-17

Dynamic Scalable Architecture
 advantages 1-8
 description of 1-8

E

Encryption, of password 5-14

Enterprise Replication 5-37
 overview of 2-6
 See also *Guide to Informix Enterprise
 Replication.*

Environment variable
 DBCENTURY 1-17
 DBSPACETEMP 15-15
 for users of client
 applications 3-21
 INFORMIXCONCSMCFG 5-21
 INFORMIXDIR 3-19
 INFORMIXSHMBASE 12-11
 INFORMIXSQLHOSTS 5-23
 ONCONFIG 3-11
 PATH 3-19
 PDQPRIORITY 16-22
Environment variables Intro-8
en_us.8859-1 locale Intro-4
Error message files Intro-13
Error messages
 for two-phase commit
 protocol 30-29
 I/O errors on a chunk 29-9
ESQL/C
 accessing smart large
 objects 14-25, 14-34
/etc/hosts file 5-15
 and client redirection 27-25
/etc/services file 5-15
 and client redirection 27-25
 multiple residency 7-10
Event alarm
 description 2-14
Event Viewer, Windows NT 2-15
Example
 DBSERVERALIASES and sqlhosts
 file 5-48
 DBSERVERALIASES
 parameter 5-47
 /etc/services file entry 5-16
 how page cleaning begins 12-40
 IPX/SPX connection 5-53
 local-loopback connection 5-51
 multiple connection types 5-53
 shared-memory connection 5-50
 TCP/IP connection 5-52
Exchange 16-15
Exclusive lock (buffer), description
 of 12-36
Explicit temporary table 14-40
Expression-based fragmentation
 scheme 16-7

Extension virtual processor. *See*
 User-defined virtual processor.
Extent
 description 14-16
 how database server
 allocates 14-17
 initial size 14-16, 14-30
 key concepts concerning 14-17
 monitoring 15-53
 next-extent size 14-16, 14-30
 purpose of 14-16
 relationship to chunk 14-17
 sbspace 14-14, 14-29
 size for sbspaces 14-29
 size for smart large object 14-14,
 14-29
 structure 14-16, 14-29
External backup and restore 1-15
 See also Backup and Restore Guide
 and *Administrator's Reference*.
External space. *See* Extspace.
External tables
 temporary table restriction 14-39
Extspace
 creating 15-37
 description of 1-23
 dropping with onspaces 15-38

F

Fast recovery
 description of 1-15, 23-14
 details of process 23-16, 23-20
 effects of buffered logging 23-16
 how database server detects need
 for 23-15
 mentioned 8-4, 9-9, 17-4
 no logging 23-16
 purpose of 23-15
 sbspaces 14-25
 when needed 23-15
 when occurs 23-14

Fault tolerance
 backup verification 1-14
 backups, and 1-14
 data replication 1-16, 27-4, 27-6
 fast recovery 1-15, 23-14
 mirroring 1-16
Feature icons Intro-10
Features of this product,
 new Intro-5
File
 configuration 3-14
 config.arc 7-11
 connectivity configuration 5-14
 cooked 1-12
 hosts.equiv 5-18
 network security 5-18
 NTFS 1-12, 14-6
 oncfg_servername.servernum
 9-10
 onconfig, during initialization 9-6
 oper_deflt.arc 7-11
 permissions 15-9
 sqlhosts 3-14
File I/O. *See* Disk I/O.
Find Error utility Intro-13
finderr utility Intro-13
FLRU queues
 and reading a page from
 disk 12-43
 and releasing buffer 12-44
 description of 12-37
 See also LRU queues.
Flushing
 buffers 12-45
 data-replication buffer, maximum
 interval 4-17
 of before-images 12-46
Forced residency
 initialization 9-10
Forcing a checkpoint. *See*
 Checkpoint.
Foreground write
 and before-image 12-46
 description of 12-51
 monitoring 12-51, 13-26
Formula
 size of physical log 21-6
Forward pointer
 how it can become invalid 23-13

Fragment
 attaching 16-5, 16-10
 creating fragmented tables 16-10
 description of 16-3
 dropping 12-21
 monitoring disk usage 15-53,
 15-56
 monitoring I/O requests
 for 15-48
 skipping 15-40
 selected fragments 15-42
 unavailable fragments 15-42
See also Fragmentation.

Fragmentation
 and memory-resident
 tables 12-21
 and mirroring 14-54
 description of 16-3
 distribution schemes for 16-6
 for improved concurrency 14-53
 goals of 16-5, 16-8
 of smart large objects 16-7
 of temporary tables 16-9
 skipping inaccessible
 fragments 15-38
 strategy 16-6
 with PDQ 16-5
See also Fragment.

Free list. *See* Chunk free list.

Free-map page
 role in blobspace logging 19-28

Full checkpoint
 description 23-5
 emptying the physical log 23-11
 events that initiate 23-7
 flushing buffer pool 23-6, 23-11
 forcing 23-8
 last available log 19-17
 oldest update 23-8

Fuzzy checkpoint
 and logical-log buffer 12-22
 and physical-log buffer 12-47
 definition of 23-4
 description 23-5
 emptying the physical log 23-11
 fast recovery 23-20 to 23-24
 flushing buffer pool 23-6, 23-10
 flushing buffers 19-31

flushing the logical-log
 buffer 19-31
 forcing 23-9
 how it works 23-9 to 23-11
 oldest update 19-17
 physical logging 21-7 to 21-11
 Fuzzy operations
 buffer pool 23-10
 description 23-5
 physical log 19-30

G

Gateway, Informix, in
 heterogeneous commit 30-38
 gcore
 file 29-7
 Global Language Support
 (GLS) Intro-4, 3-12
 Global pool, description of 12-33
 Global transaction identification
 number 31-7, 31-8
 GRANT FRAGMENT
 statement 27-33
 GRANT statement 27-33
 Group, parallel processing of 10-11

H

Hash table
 to buffer table 12-27
 Heaps 12-32
 Heterogeneous
 commit 30-38 to 30-45
 Heuristic decision
 result from independent
 action 30-20
 types of 30-21
 Heuristic end-transaction

conditions for 30-26
 description of 30-26
 determining effects on
 transaction 31-4
 illustration, including logical-log
 records 30-35
 messages returned by 30-28

results of 30-28
 when necessary 30-26
 Heuristic rollback
 conditions resulting in 30-22
 description of 30-22
 illustration, including logical-log
 records 30-33
 indications that rollback
 occurred 31-5
 results of 30-23
 High availability
 description of 26-3
 sharing hard disks in a
 cluster 26-4
 High-availability data replication.
See Data replication.
 HKEY_LOCAL_MACHINE
 registry 3-14
 hostname field
 multiple network interface
 cards 10-38
 syntax rules 5-31
 using IP addresses 5-42
 wildcard addressing 5-43
 with IPX/SPX 5-32
 with shared memory 5-32, 5-34
 hosts file 5-16
 multiple residency 7-10
 on UNIX 5-15
 hosts.equiv file 5-18
 Hot site backup. *See* Data
 replication.

I

Icons
 feature Intro-10
 Important Intro-10
 platform Intro-10
 product Intro-10
 Tip Intro-10
 Warning Intro-10
 IDX_RA_PAGES parameter
 description of 12-41
 Ill-behaved user-defined
 routine 10-25
 Impersonate, client 5-20

Important paragraphs, icon
for Intro-10

Inconsistency information
how to detect 29-3

Incremental backup
description of 2-8
sbspaces 19-33

Index
access methods 1-25
parallel building of 10-10
R-tree 21-6
tblspace 14-42

Industry standards, compliance
with Intro-15

.informix file
multiple residency 7-12

Informix recommendations
on allocation of disk space 14-10
on consistency checking 29-3
on mirroring the physical
log 21-9

Informix Storage Manager
(ISM) 3-22, 26-6

Informix-Admin group 8-6

INFORMIXCONCSMCFG
environment variable 5-21

INFORMIXDIR
environment variable
in shutdown script 3-20
in startup script 3-19
multiple residency startup
script 7-12

\$INFORMIXDIR/etc/sqlhosts. *See*
sqlhosts file.

INFORMIXDIR/bin
directory Intro-5

INFORMIXSERVER environment
variable
and dserver group 5-39
multiple residency startup
script 7-12
multiple versions of the database
server 3-20
use in client redirection 27-27
with multiple residency 7-12

INFORMIXSHMBASE
environment variable 12-11

INFORMIXSQLHOSTS
environment variable 5-23, 5-25

INFORMIXSTACKSIZE
environment variable 12-31

informix.rc environment file
multiple residency 7-12

Inheritance 1-22
type and table, description
of 1-22
type and table, mentioned 1-20

Initial configuration
creating storage spaces 3-22
disk layout 14-48
guidelines for root dbspace 14-22

Initial-extent size 14-16

Initialization
checkpoint 9-9
configuration changes 9-9
configuration files 9-6
control returned to user 9-11
conversion of internal files 9-9
disk space 3-17, 9-3, 9-8, 15-6
for multiple residency 7-10
fast recovery 9-9
forced residency 9-10
message log 9-11
oncfg_servername.servernum
file 9-10
onconfig file 9-6
shared memory 9-3
SMI tables 9-11
steps in 9-5
temporary tablespaces 9-10
utilities for 9-4
virtual processors 9-8

INSERT cursor 16-7

Installation, definition of 3-9

installserver script 3-10

Integrity, data. *See* Consistency
checking.

Integrity, of communication
messages 5-13

Interprocess communications
in nettype field 5-30
shared memory for 12-5

IP address
how to find 5-42
use in hostname field 5-42

ipcshm protocol and
communications portion
(shared memory) 12-34

IPC. *See* Interprocess
communications.

IPX/SPX
in hostname field 5-53
in servicename field 5-34
multiple residency 7-10
service, definition of 5-34
sqlhosts entry 5-53

ISO 8859-1 code set Intro-4

Isolation level
Committed Read and blob time
stamps 23-13
Dirty Read and TEXT or BYTE
data 23-13

I/O. *See* Disk I/O.

J

Java user-defined routine
mentioned 1-24

Java virtual processor 10-27

Join, parallel processing of 10-10

K

KAIO thread 10-27, 10-30

Keep-alive option, in options
field 5-39

Kernel asynchronous I/O
description of 10-30
nonlogging disk I/O 10-27

Key value
for shared memory 12-13

L

Large objects
storage, mentioned 1-18

Latch
and wait queue 10-18
monitoring statistics and
use 13-28
mutex 10-19

Latch. *See* *mutex*. 12-35

Level-0 backup
use in consistency checking 29-8

- Levels, backup
 - description of 2-8
 - sbspaces 19-33
- Light scans
 - mentioned 12-30
- Lightweight I/O 12-20, 14-32
- Lightweight processes 10-4
- Limits
 - chunk size 15-19
 - number of chunks 14-6
- Links, creating 15-12
- LIO virtual processors
 - description of 10-28
 - how many 10-29
- Listen threads
 - and multiple interface cards 10-38
 - description of 10-34
- Load balancing
 - as performance goal 14-48
 - done by virtual processors 10-9
 - of critical media 14-50
 - through use of DBSPACETEMP 14-50
- Loading data
 - methods 15-28
- Local loopback
 - connection 5-13, 10-32
 - example 5-51
 - restriction 5-13
- Locale Intro-4
 - default Intro-4
 - en_us.8859-1 Intro-4
- Lock
 - and wait queue 10-18
 - description of 12-34
 - types 12-35
- Lock table
 - configuration 12-24
 - contents of 12-24
 - description 12-24
- Lock-access level, of a buffer 12-43
- Locking
 - Dirty Read isolation level 27-34
 - sbspaces 14-33
 - smart large objects 14-25
 - when occurs 12-43
 - when released 12-44
- LOGBUFF parameter
 - and logical log buffers 12-22
 - smart large objects and 12-58
 - smart large objects, and 12-58
- LOGFILES parameter
 - changing 20-10
 - use in logical-log size determination 19-8
- Logging
 - activity that is always logged 17-7
 - and R-tree indexes 21-6
 - database server processes requiring 17-4
 - definition of transaction logging 17-6
 - effect of buffering on logical log fill rate 19-18
 - monitoring activity 22-6
 - physical logging
 - description of 21-3
 - process of 21-9
 - purpose of 21-4
 - sizing guidelines for 21-6
 - suppression in temporary dbspaces 14-23
 - process for blobspace data 19-32
 - process for dbspace data 19-29
 - role in data replication 27-9
 - role of blobspace free-map page 19-32
 - sbspaces 14-33, 19-32
 - smart large objects 12-58
 - suppression for implicit tables 14-23
 - when to buffer transaction logging 17-13
 - when to use logging tables 17-11
 - when to use transaction logging 17-12
 - with TEXT and BYTE data 17-9
 - See also* Database-logging status.
- LOGGING tag, onspaces 14-33
- Logical consistency, description of 23-16, 23-20
- Logical log
 - administration tasks for
 - blobspaces 19-26
 - backing up 19-24
 - configuration parameters 20-10
 - description of 12-21, 17-4, 19-3
 - determining disk space
 - allocated 19-8, 19-9
 - how to prevent logs full 19-24
 - monitoring for fullness using onstat 20-16
 - optimal storage of 14-50, 14-51
 - preserving space in 19-24
 - purpose of 1-14
 - setting high-water marks 19-20
 - size, guidelines 19-7
 - size, performance
 - considerations 19-5
 - types of records 12-21
 - See also* Logical-log buffer; Logical-log file.
- Logical recovery
 - role in data replication 27-10
- Logical units of storage
 - description 14-19
 - list of 14-4
- Logical volume manager (LVM)
 - description of 14-61
 - mirroring alternative 24-6
- Logical-log backup
 - and checkpoints 23-11
 - description of 2-8
- Logical-log buffer
 - and checkpoints 12-22
 - and logical-log records 12-52
 - description 12-21
 - description of 12-21
 - flushing 12-52
 - flushing for nonlogging databases 12-54
 - flushing when a checkpoint occurs 12-54
 - flushing when no before-image 12-55
 - flushing with unbuffered logging 12-54
 - fuzzy checkpoints 19-31
 - monitoring 22-7
 - synchronizing flushing 12-46
 - when flushed to disk 12-52, 19-31
 - when it becomes full 12-53

Logical-log file
 adding a log file
 using ON-Monitor 20-5
 using onparams 20-5, 20-9
 allocating disk space for 19-5
 backing up 15-13
 backing up after adding, moving,
 or deleting 20-4
 backup
 effect on performance 19-5
 goals of 19-14
 to free deleted blobpages 19-26
 changing the size of 20-9
 consequences of not freeing 19-16
 description of 17-4, 19-4
 dropping a log file
 using ON-Monitor 20-7
 using onparams 20-6
 file status 19-12
 full checkpoint to free 23-8
 how to free 20-13
 how to switch 20-15
 I/O to 10-27
 location 19-10
 logid number 19-11
 mirroring a dbspace that contains
 a file 24-8
 moving to another dbspace 20-7
 number of files 19-8
 rate at which files fill 19-18
 relationship between unique ID
 and logid 19-11
 role in fast recovery 23-17,
 23-18 to 23-19, 23-23 to 23-24
 size 19-7
 status flags 19-12
 switching to activate blobpage
 chunks 19-26
 switching to activate
 blobpages 19-26
 unique ID number 19-11
 when database server tries to free
 files 19-15
 when freed for reuse 19-15
 See also Logical log.
 Logical-log I/O virtual
 processors 10-28

Logical-log record
 database server processes
 requiring 17-4
 flushing under two-phase commit
 protocol 30-32
 involved in two-phase commit
 protocol 30-30
 role in two-phase commit
 protocol 30-6
 SQL statements that
 generate 17-8
 when written to logical-log
 buffer 19-31
 logid 19-11
 Logs-Full High-Water Mark 19-24
 LOGSIZE parameter
 changing 20-10
 use in logical-log size
 determination 19-8
 LOGSMAX parameter
 changing 20-12
 mentioned 19-8
 Long transaction
 consequences of 19-16
 description of 19-17
 mentioned within two-phase
 commit discussion 30-19,
 30-23, 30-26
 preventing development of 19-18
 LRU queues
 and buffer pool
 management 12-37
 components 12-36
 composition of 12-36
 description of 12-36
 FLRU queues 12-37
 how database server selects 12-38
 MLRU queues 12-37
 ordering of 12-37
 rationale for ordering 12-37
 reason for multiple 12-38
 LRU write
 description of 12-51
 monitoring 12-26
 triggering of 12-51
 who performs 12-51
 LRUS parameter
 configuration 12-38
 description of 12-36

LRU_MAX_DIRTY parameter
 and LRU_MIN_DIRTY
 parameter 12-36
 description 12-39
 example of use 12-40
 how to calculate value 12-40
 role in buffer-pool
 management 12-40
 LRU_MIN_DIRTY parameter
 and LRU_MAX_DIRTY
 parameter 12-36
 default value 12-40
 example of use 12-40
 how to calculate value 12-40
 role in buffer pool
 management 12-40
 when tested 12-50
 LTXEHWM parameter
 and physical log 21-8
 changing 20-12
 role in heuristic rollback 30-22
 role in preventing long
 transactions 19-21
 LTXHWM parameter
 changing 20-12
 flushing of logical-log
 buffer 12-53
 role in heuristic rollback 30-22
 role in preventing long
 transactions 19-20

M

Machine notes Intro-14
 Main_loop() thread 23-10
 role in two-phase commit
 recovery 30-12
 Management of data 14-8
 Manual recovery
 deciding if action needed 31-9
 determining if data
 inconsistent 31-6
 example of 31-10
 obtaining information from
 logical-log files 31-7
 procedure to determine if
 necessary 31-3
 use of GTRID 31-7

Mapping, bad sector 1-16, 15-63

Media failure

detecting 24-10

recovering from 24-5

restoring data 1-14

Memory

64-bit addressing 12-59

adding a segment 13-18

Memory Grant Manager (MGM)

description 16-21

See also Performance Guide.

Memory-resident tables

adding a fragment to 12-21

description 12-20

Memory. *See* Shared memory.

mentioned 12-7

Message file for error

messages Intro-13

Message log

and data corruption 29-9

description of 2-13

during initialization 9-11

Metadata 14-26

allocating 15-24

and chunks 15-27

and dbspace table 12-28

calculating area 14-34

calculating size 15-25

definition 14-26

description 12-58

dropping sbspace chunks 15-31

logging 21-5

sbspace logging 19-32

meter 5

MGM. *See* Performance Guide.

Microsoft Cluster Services (MSCS)

administration 26-6

definition of 26-3

description of 26-3

multiple residency 26-5

usage of 26-4

Migration. *See* Informix Migration Guide.

Mirror chunk

adding 25-11

changing status of 25-8

creating 25-7

disk reads from 24-10

disk writes to 24-9

recovering 24-11, 25-8

structure 24-11

Mirror dbspace

creating 15-10

root dbspace 14-22

MIRROR parameter

changing 25-4

description of 25-4

initial configuration value 25-4

Mirroring

activity during processing 24-9

alternatives 24-6

and chunk table 12-27

and multiple residency 7-9

asynchronous write requests 24-9

benefits of 24-4

changing chunk status 25-8

costs of 24-5

creating mirror chunks 25-7

description of 1-16, 24-4

detecting media failures 24-10

during system initialization 25-8

enabling 25-4

ending 25-9, 25-13

if the dbspace holds logical-log files 24-8

network restriction 24-4

recommended disk layout 14-54

recovering a chunk 25-8

recovery activity 24-9

split reads 24-10

starting 25-3, 25-7

status flags 24-8

steps required 25-3

what happens during

processing 24-9

when mirroring begins 24-7

when mirroring ends 24-11

MIRROROFFSET parameter

root dbspace 14-22

when needed 15-8

MIRRORPATH parameter

and multiple residency 6-5

mentioned 14-22

multiple residency 7-9

Miscellaneous (MSC) virtual

processor 10-40

Mixed transaction result 30-25

MLRU queues

and flushing of regular

buffers 12-46

and LRU_MIN_DIRTY

parameter 12-40

description of 12-37

end of cleaning 12-40

how buffer is placed 12-38

how to end page-cleaning 12-40

limiting number of pages 12-39

role in buffer modification 12-45

See also LRU queues.

Mode

description of 8-3

graceful shutdown 8-7

immediate shutdown 8-8

off-line from any mode 8-9

off-line to on-line 8-6

off-line to quiescent 8-6

on-line to quiescent,

gracefully 8-7

on-line to quiescent,

immediately 8-8

quiescent to on-line 8-7

reinitializing shared memory 8-6

taking off-line 8-9

MODE ANSI keywords, and

database logging status 17-13

Monitoring

memory size 13-6

sbspaces 14-34

Monitoring database server

blob space storage

considerations 15-34

buffer-pool activity 13-25

buffers 13-21

checkpoints 23-24

chunks 15-46

configuration parameter

values 3-23

databases 15-43, 18-11

data-replication status 28-24

disk I/O queues 10-31

extents 15-53

fragment load 15-48

fragmentation disk use 15-53,

15-56

latches 13-19, 13-28

length of disk I/O queues 10-31

- logging status 18-11
- logical-log buffers 22-7
- logical-log files 20-16
- physical-log buffer 12-23, 22-7
- physical-log file 22-6
- profile of activity 13-20
- shared memory 13-19
- shared-memory segments 13-19
- simple large objects in
 - dbspaces 15-61
- sources of information 2-13
- user threads 12-30
- using oncheck 2-17
- using ON-Monitor 2-15
- using onperf 2-17
- using onstat 2-16
- using SMI tables 2-15
- virtual processors 11-12
- MSGPATH parameter
 - and multiple residency 6-5, 7-9
- Multiple concurrent threads (MCT) 10-13
- Multiple connection types
 - example 5-53
 - in sqlhosts 5-48*See also* Connection.
- Multiple instances of database
 - server on same computer 6-3
- Multiple network interface
 - cards 10-38
- Multiple residency
 - and blobspace 7-10
 - and chunk assignment 7-10
 - and dbspaces 7-10
 - and multiple binaries,
 - warning 7-4
 - archiving 7-11
 - backups 7-11
 - benefits of 6-3
 - configuration file 6-3
 - configuration parameters 6-5
 - DBSERVERNAME parameter 7-8
 - definition of 3-9, 5-55, 6-3
 - editing the ONCONFIG file 7-8
 - example 5-55
 - hosts and services files 7-10
 - how it functions 6-4
 - INFORMIXSERVER environment
 - variable 7-12

- informix.rc & .informix files 7-12
- initializing disk space 7-10
- IPX/SPX 7-10
- MIRRORPATH parameter 7-9
- MSGPATH parameter 7-9
- ONCONFIG environment
 - variable 6-5
- planning for 7-7
- requirements 7-4
- ROOTNAME parameter 7-9
- ROOTPATH parameter 7-9
- SERVERNUM parameter 7-8
- sqlhosts 7-10
- startup script 7-12
- to isolate applications 6-4
- use for testing 6-4
- Windows NT 26-5
- Multiplexed connection 5-7
 - description of 5-7
 - limitations 5-8
- Multiprocessor computer
 - AFF_SPROC parameter 4-15
 - MULTIPROCESSOR
 - parameter 10-21
 - processor affinity 10-12
 - MULTIPROCESSOR parameter
 - description of 10-21
 - for single-processor
 - computer 10-21
- Multithreaded database server. *See* Dynamic Scalable Architecture.
- Multithreaded processes,
 - description of 1-9, 10-4
- Multithreading, use of OS
 - resources 10-10
- Mutex
 - description of 10-19, 12-35
 - on buffer-table hash table 12-43
 - synchronization 10-19
 - when used 12-35

N

- Named row type, definition of 1-22
- Named-pipe connection
 - description of 5-12
 - mentioned 5-9
 - platforms supported 5-9

- Names, storage spaces 15-15
- nettype field
 - format of 5-29
 - summary of values 5-31
 - syntax of 5-29
 - use of interface type 5-52
- NETTYPE parameter 5-48
 - and multiple network
 - addresses 10-38
 - for multiplexed connection 5-7
 - mentioned 5-48
 - ON-Monitor screen entries 11-7
 - poll threads 10-32
 - purpose of 5-48
 - role in specifying a protocol 10-32
 - vp class entry 10-33
- NetWare file server 5-32
- Network communication
 - using IPX/SPX 5-32, 5-34, 5-53
 - using TCP/IP 5-31, 5-32
- Network connection
 - how implemented 10-34
 - types of 10-32
- Network Information Service 5-16
- Network interface cards
 - and listen threads 10-38
 - sqlhosts file 10-38
 - using multiple 10-38
- Network protocols,
 - specifying 10-32
- Network security
 - files 5-18
 - hosts.equiv 5-18
 - .netrc file 5-19
- Network virtual processors
 - and poll threads 10-32
 - description of 10-32
 - how many 10-33
- New features of this
 - product Intro-5
- Next-extent size 14-16
- NIS servers, effect on /etc/hosts
 - and /etc/services 5-16
- NOAGE parameter
 - purpose of 10-22
- Nonfuzzy operations
 - description 23-6
 - flushing buffer pool 23-6

flushing buffers 19-31, 23-9, 23-10
 physical log 19-30
 Non-Informix databases 1-18
 NTFS files 1-12, 14-6
 Null file
 for creating a chunk 15-10
 NUMAIOVPS parameter
 purpose of 10-31
 NUMCPUVPS parameter
 and poll threads 10-33
 purpose of 10-20

O

Object-relational database-
 management system 1-5, 1-6,
 1-10, 1-20
 Offset
 definition of 14-10
 purpose of 15-7
 use in prevention of overwriting
 partition information 14-10
 use in subdividing
 partitions 14-10
 when needed 15-7
 OFF_RECOVERY_THREADS
 configuration parameter 4-16
 Oldest update, freeing logical-log
 file 19-15
 OLTP application
 characteristics of 1-6, 16-17
 definition of 16-17
 uses of 1-6, 16-17
 OLTP applications
 See also PDQ (parallel database
 query).
 ON-Archive
 backing up logical-log files 19-16
 estimating disk space for 14-47
 modifying database logging
 status 18-6
 use in starting data
 replication 28-11
 ON-Bar
 backing up logical-log files 19-14
 oncatlgr utility
 multiple residency 7-11

oncfg_servername.servernum
 file 9-10
 oncheck utility
 blobpage information 15-35
 check reserved pages 22-8
 obtaining blobpage
 information 15-60
 obtaining blobpage storage
 statistics 15-59
 obtaining chunk
 information 15-48, 15-49
 obtaining configuration
 information 3-23
 obtaining extent
 information 15-53
 obtaining fragmentation
 information 15-54
 obtaining logical-log
 information 20-18
 obtaining tblspace
 information 15-55
 options
 -pB 15-34
 -pr option 21-6
 use in consistency checking 29-4
 ONCONFIG configuration file
 during initialization 9-6
 editing with multiple
 residency 7-8
 mentioned 5-46
 multiple residency 5-55, 7-8
 parameters 5-47
 ONCONFIG environment
 variable 3-11
 changes for multiple
 residency 7-8
 multiple database servers 3-20
 multiple residency startup
 script 7-12
 use with multiple residency 6-5
 ONCONFIG file parameters. *See*
 Configuration parameter.
 ONCONFIG parameter, and
 multiple residency 7-7
 ondblog utility
 modifying database logging
 status 18-5

oninit utility
 -p option 9-10, 14-39
 temporary tables 14-39
 On-line help Intro-13
 On-line manuals Intro-12
 On-line transaction processing. *See*
 OLTP applications.
 onload utility. *See Informix*
 Migration Guide.
 onlog utility
 reconstructing a global
 transaction 31-7
 onmode utility
 -a option 13-18
 adding a segment 13-18
 -c block option 2-11
 changing shared-memory
 residency 13-17
 dropping CPU virtual
 processors 11-11
 forcing a full checkpoint 23-9
 forcing a fuzzy checkpoint 23-9
 freeing a logical-log file 20-14,
 20-15
 killing a participant thread 30-19
 killing a session 30-22
 killing a transaction 30-19, 30-27,
 31-4
 -O option 14-18
 setting database server
 type 28-11, 28-22
 switching logical-log files 20-15
 user thread servicing requests
 from 10-5
 ON-Monitor
 adding a chunk 15-19
 adding a logical-log file 20-5
 adding mirror chunks 25-11
 creating blobspaces 15-23
 creating dbspaces 15-17
 dropping a logical-log file 20-7
 dropping storage spaces 15-33
 recovering a chunk 25-12
 setting performance
 options 13-14
 setting shared memory
 parameters 13-9, 13-12
 setting virtual processor
 parameters 11-6

- starting mirroring 25-10
- taking a chunk down 25-12
- onparams utility
 - adding a logical-log file 20-5
 - changing physical-log location 22-5
 - changing physical-log size 22-5
 - dropping a logical-log file 20-6
- onperf utility 2-17
 - description of 1-17
- onsmsync utility. *See Backup and Restore Guide.*
- onspaces utility
 - a option 15-19
 - adding a mirror chunk 25-11
 - c -b option 15-23
 - c -d option 15-16
 - c -S option 14-27, 15-24
 - c -t option 15-16
 - c -x option 15-37
 - ch option 14-29, 14-31, 14-33
 - d option 15-33, 15-38
 - Df tags 14-29, 14-31
 - ending mirroring 25-13
 - example modifying DATASKIP 15-39
 - f option 15-39
 - modifying DATASKIP parameter 15-39
 - recovering a down chunk 25-12
 - starting mirroring 25-10
 - taking a chunk down 25-11
 - U option 15-26
- onstat utility
 - and CPU virtual processors 10-20
 - g seg option 13-6
 - monitoring blobspace 15-57
 - monitoring buffer use 13-21, 13-22, 13-23
 - monitoring buffer-pool 13-25, 13-26
 - monitoring checkpoints 23-24
 - monitoring chunk status 15-46
 - monitoring configuration 3-23
 - monitoring data replication 28-24
 - monitoring database server profile 13-20
 - monitoring fragment load 15-48
 - monitoring latches 13-28, 13-29

- monitoring log buffers 22-7
- monitoring logical-log files 20-17
- monitoring physical log 22-7
- monitoring shared memory 13-19
- monitoring virtual processors 11-13
- tracking a global transaction 30-28
- ontape utility
 - backing up logical-log files 19-16
 - modifying database logging status 18-9
 - use in starting data replication 28-11
- onunload utility. *See Informix Migration Guide.*
- ON_RECOVERY_THREADS
 - configuration parameter 4-16
- ON_RECVRY_THREADS
 - parameter role in data replication 27-14
- Opaque data types
 - mentioned 1-21
- Opaque type, description of 1-21
- OPCACHEMAX parameter
 - configuring memory 15-58
 - description of 4-19
- Operating system, scheduled jobs 3-21
- Operating-system files. *See Cooked file space.*
- Operating-system parameters
 - description of 13-3
- OPTCOMPIND
 - description 16-22
- OPTCOMPIND environment variable
 - See also* OPTCOMPIND parameter.
- Optical Subsystem memory cache
 - allocation 15-58
 - availability of memory 15-58
 - described 15-58
 - kilobytes of TEXT and BYTE data written 15-58, 15-59
 - number of objects written 15-58, 15-59
 - session ID for user 15-59
 - size 15-58

- specifying size of 4-19
- user ID of client 15-59
- Optical (OPT) virtual processor 10-39
- options field
 - buffer-size option 5-35
 - keep-alive option 5-39
 - security option 5-40
 - syntax rules 5-34
- ORDBMS. *See* Object-relational database management system.

P

- Page
 - description 14-11
 - determining database server page size 15-22
 - least-recently used 12-37
 - locating in shared memory 12-43
 - most-recently used 12-37
 - relationship to chunk 14-11
- Page size
 - shown in ON-Monitor 13-10
- Page-cleaner table
 - description of 12-28
 - number of entries 12-29
- Page-cleaner threads
 - alerted during foreground write 12-51
 - description of 12-45
 - flushing buffer pool 12-46
 - flushing of regular buffers 12-46
 - monitoring 12-29
 - role in chunk write 12-52
 - sleeping forever 12-50
- PAGE_CONFIG reserved page
 - mentioned 9-7, 9-9
- PAGE_PZERO reserved page
 - mentioned 9-8
- Parallel database query. *See* PDQ.
- Parallel processing
 - mentioned 1-12
 - MGM control of resources 16-21
 - virtual processors 10-10

- Parallelism
 - degree of, definition 16-12
- Parameters
 - used with PDQ 16-21
- parameters, setting
 - with a text editor 13-13
- Participant database server
 - automatic recovery 30-14
 - description of 30-5
- Password encryption 5-14, 5-22
- PATH environment variable
 - in shutdown script 3-20
 - in startup script 3-19
 - multiple residency startup script 7-12
- PC_HASHSIZE configuration
 - parameter 4-11, 13-13
- PC_POOLSIZ configuration
 - parameter 4-11, 13-13
- PDQ (parallel database query)
 - degree of parallelism 16-12
 - description of 16-11
 - DS_TOTAL_MEMORY
 - parameter 4-13
 - effect of table fragmentation 16-3
 - parameters used to control 16-21
 - PDQPRIORITY environment variable 16-22
 - principal components 16-12
 - resource allocation 16-21
 - SET PDQPRIORITY
 - statement 16-22
 - used with fragmentation 16-5
 - uses of 16-3
 - when to use 16-16
- PDQPRIORITY environment variable
 - values of 16-22
- Performance 13-13
 - and resident shared-memory 12-17
 - and shared memory 12-5
 - and yielding functions 10-14
 - effect of read-ahead 12-41
 - effects of VP-controlled context switching 10-10
 - how frequently buffers are flushed 12-36
 - of CPU virtual processors 10-20
 - parameters, setting
 - with ON-Monitor 13-14
 - shared-memory connection 5-11
 - See also Performance Guide.*
- Performance tuning
 - and extent size 14-55
 - and foreground writes 12-51
 - and logical volume managers 14-61
 - blobspace blobpage size 15-33
 - disk-layout guidelines 14-48
 - logical-log size 19-5
 - LRU write 12-51
 - mechanisms 1-11
 - minimizing disk-head movement 14-55
 - moving the physical log 22-4
 - sample disk layout for optimal performance 14-57
 - spreading data across multiple disks 14-61
 - tuning amount of data logged 21-5
- Permissions, file 15-9
- PHYSBUFF parameter
 - and physical-log buffers 12-23
- PHYSDBS parameter
 - changing size and location 22-5
 - where located 21-9
- PHYSFILE parameter
 - changing size and location 22-5
- Physical consistency
 - description of 23-16
 - during fast recover 23-20
- Physical consistency, description of 23-5
- Physical log
 - and virtual processors 10-27
 - backing up 15-13
 - becoming full 21-7
 - before-image contents 21-5
 - buffer 21-9
 - changing size and location
 - possible methods 22-3
 - rationale 22-4
 - restrictions 22-4
 - using a text editor 22-5
 - using ON-Monitor 22-6
 - using onparams 22-5
 - contiguous space 22-4
 - description of 21-3
 - ensuring does not become full 21-7
 - flushing of buffer 21-10
 - how emptied 21-11
 - I/O, virtual processors 10-29
 - managing 22-3
 - monitoring 22-6
 - optimal storage of 14-50, 14-51
 - reinitialize shared memory 22-3
 - role in fast recovery 23-15, 23-17, 23-20, 23-21
 - scenario for filling 21-8
 - sizing guidelines 21-6, 21-7
 - where located 21-9
- Physical logging
 - activity logged 21-5
 - and backups 21-4
 - and blobspace blobpages 21-5
 - and data buffer 21-10
 - and fast recovery 21-4
 - description of 21-3
 - details of logging process 21-9
 - fuzzy checkpoints 21-7 to 21-11
 - purpose of 21-4
 - smart large objects 21-5
- Physical units of storage, list of 14-3
- Physical-log buffer
 - amount written 12-49
 - and checkpoints 12-47, 21-11
 - description of 12-23
 - events that prompt flushing 12-47
 - flushing of 12-46, 21-10
 - mentioned 12-48
 - monitoring 22-7
 - number of 12-23
 - PHYSBUFF parameter 12-23
 - role in dbspace logging 19-30, 21-10
 - when it becomes full 12-48
- PIO virtual processors
 - description of 10-29
 - how many 10-29
- Planning for resources 3-4
- Platform icons Intro-10

Point-in-time recovery. *See Backup and Restore Guide or Archive and Backup Guide.*

Poll threads

and message queues 12-33

DBSERVERNAME

parameter 10-33

description of 10-34

how many 10-33

multiple for a protocol 10-32

nettype entry 10-32

on CPU or network virtual

processors 10-33

Post-decision phase 30-6, 30-9

Precommit phase 30-6

Presumed-abort

optimization 30-10, 30-17

Primary database server 27-5

Printed manuals Intro-13

Priorities for disk I/O 10-28

Priority aging

description of 10-22

preventing 10-22

Private installation

installserver script 3-10

purpose 3-10

Privileges

on databases and tables 1-17

Processes

attaching to shared

memory 12-11

comparison to threads 10-3

DSA versus dual process

architecture 10-10

Processor affinity

and AFF_SPROC parameter 4-15

description of 10-12

using 10-23

Product icons Intro-10

Profile statistics 13-20

Program counter and thread

data 12-31

Program group

Documentation notes Intro-14

Release notes Intro-14

Protocol, specifying 10-32

PSORT_DBTEMP environment

variable, creating temporary

implicit tables 14-41

PSWDSCSM. *See* Password encryption.

Q

Queues

description of 10-16

disk I/O 10-31

ready 10-17

sleep 10-17

wait 10-18

R

RAID. *See* Redundant array of inexpensive disks.

Raw device 14-7

and character-special

interface 14-7

definition of 14-7

Raw disk space 14-7

comparison to cooked space 14-8

how to allocate, UNIX 15-10

how to allocate, Windows

NT 15-12

on Windows NT 14-6

rationale for using 14-8

steps for allocating 14-8

RA_PAGES parameter

purpose of 12-41

reading a page from disk 12-43

RA_THRESHOLD parameter

description of 12-41

RDBMS. *See* Relational database

management system.

Read-ahead

description of 12-41

IDX_RA_PAGES

parameter 12-41

RA_THRESHOLD

parameter 12-41

when it occurs 12-43

when used 12-41

Read-only mode, description of 8-4

Ready queue

description of 10-17

moving a thread to 10-17, 10-18

Reception buffer 27-10

Recovery

by two-phase commit

protocol 30-10

fast, description of 23-14

from media failure 24-5

Microsoft clusters 26-3

parallel processing of 10-10

Recovery mode, description of 8-4

Redundant array of inexpensive

disks (RAID), mirroring

alternative 24-7

Referential constraints 17-5

regedt32 program

and sqlhosts registry 5-25

registry

defining multiple network

addresses 10-37

Regular buffers

events that prompt flushing 12-46

monitoring status of 12-19

Related reading Intro-15

Relational database management

system (RDBMS) 1-20, 1-22

Release notes Intro-14

Remote client 5-18

Remote hosts 5-18

Replication server. *See* Data

replication.

Reserved pages

estimating disk space for 14-47

optimal storage 14-50

Residency. *See* Multiple residency.

RESIDENT parameter

during initialization 9-10

mentioned 13-17

Resident shared memory 9-7, 12-18

conditions for 12-17

contents of 12-18

description 12-17, 12-18

internal tables 12-26

Resource allocation

effects of PDQ 16-21

Resource planning 3-5

Restore

data 1-14

external 1-15, 2-11

REVOKE FRAGMENT

statement 27-33

REVOKE statement 27-33

rofferr utility Intro-13

Roll back

- in fast recovery 23-19, 23-23
- mentioned 1-15

Roll forward

- in fast recovery 23-18, 23-23
- mentioned 1-15

Root dbspace

- and temporary tables 14-22, 14-41
- calculating size of 14-45
- default location 14-22
- description of 14-22
- location of logical-log files 19-10
- mirroring 25-8

ROOTNAME parameter

- mentioned 14-22
- multiple residency 7-9

ROOTOFFSET parameter

- mentioned 14-22
- multiple residency 7-9
- when needed 15-8

ROOTPATH parameter

- and multiple residency 6-5
- mentioned 14-22
- multiple residency 7-9

Round-robin fragmentation

- distribution scheme 16-7
- smart large objects 16-7

Row type

- named 1-22

Row types 1-22

R-tree indexes

- logging 21-6
- oncheck and onlog
- options Intro-6

S

Sample-code conventions Intro-11

Sbpage

- description of 14-14
- sizing recommendations 14-14

Sbspace

- adding a chunk 14-34, 15-27
- allocating space 14-34
- altering storage
- characteristics 15-28
- backing up 14-34, 15-13, 15-25

- buffering mode 14-32
- calculating metadata 14-34
- checking consistency 14-35
- creating with onspaces 15-24
- description 1-23
- description of 14-24
- disk structures 14-35
- dropping 15-31
- dropping with onspaces 15-33
- fast recovery 14-25
- incremental backups 19-33
- last-access time 14-32
- lightweight I/O 12-20, 14-32
- locking 14-33
- logging 14-33, 19-32
- metadata 12-58, 14-26
- names 15-15
- recoverability 14-25
- sbpages 14-14
- sizing metadata 15-25
- specifying location of smart large objects 14-27
- specifying storage
- characteristics 15-24
- statistics for user-defined data 4-5
- storage characteristics 14-29
- storing smart large objects 12-58, 14-24
- tuning 14-34
- user data 14-26
- user-data area 12-58, 15-26

SBSPACENAME parameter 14-27

Scalability of database server 1-10

Scans

- of indexes 12-41
- of sequential tables 12-41
- parallel processing of 10-10

Scheduled system jobs 3-21

Scripts

- database server installation 3-10

Secondary database server 27-5

Security

- database server 1-17
- how enforced 1-17
- isolating applications 6-4
- risks with shared-memory communications 5-11

Security options, sqlhosts 5-40

Security option, in options field 5-40

Segment identifier (shared-memory) 12-14

Segment. *See* Chunk.

Semaphore, UNIX parameters 13-7

SERVERNUM parameter

- and multiple database servers 12-13
- and multiple residency 6-4, 6-5, 7-8
- description of 12-12, 12-13
- how used 12-13

servicename

- syntax rules 5-31

servicename field in sqlhosts file

- choosing an appropriate name 5-32
- with IPX/SPX 5-34
- with shared memory 5-33
- with stream pipes 5-33

services file 5-16

- multiple residency 7-10

Service, in IPX/SPX 5-34

Session

- and active tblspace 12-29
- and dictionary cache 12-32
- and locks 12-24
- and shared memory 12-30
- and UDR cache 12-33
- control block 10-13
- description of 10-13
- primary thread 12-30
- shared-memory pool 12-25
- sqlxec threads 10-5
- threads 10-5

Session control block 10-13

- description of 12-30
- shared memory 12-30

SET DATASKIP statement 15-39, 15-40

SET INDEX statement 12-20, 12-21

SET PDQPRIORITY statement 16-22

SET TABLE statement 12-20, 12-21

Setting the year 1-17

setup program

- and sqlhosts registry 5-24

- Share lock (buffer), description of 12-36
- Shared data 12-5
- Shared memory
 - allocating 12-25
 - and blobpages 12-55
 - and critical sections 23-4
 - and SERVERNUM parameter 12-12
 - and SHMBASE parameter 12-12
 - and smart large objects 12-58
 - attaching additional segments 12-13, 12-15
 - attaching to 12-11
 - buffer allocation 12-19
 - buffer hash table 12-27
 - buffer locks 12-35
 - buffer pool 12-18, 21-10
 - buffer table 12-27
 - changing residency with onmode 13-17
 - checkpoint 23-4
 - chunk table 12-27
 - communication 5-32, 5-34
 - communications portion 12-33
 - configuration 12-9, 12-25, 13-3
 - configuring for multiple residency 6-6
 - contents 12-8, 12-34
 - copying to a file 13-19
 - data-replication buffer 27-10
 - dbspace table 12-28
 - description 12-5
 - dictionary cache 12-32
 - dynamic management of 1-11
 - effect of operating-system parameters 13-3
 - first segment 12-13
 - for interprocess communication 12-5
 - global pool 12-33
 - header 12-15, 12-18
 - heaps 12-32
 - how utilities attach 12-12
 - identifier 12-13
 - initializing 9-3
 - initializing structures 9-8
 - internal tables 12-26
 - key value 12-13
 - locating a page 12-43
 - logical-log buffer 12-21
 - lower-boundary address problem 12-15
 - memory-resident tables 12-20
 - mirror chunk table 12-27
 - monitoring 13-19
 - mutex 12-35
 - operating-system segments 12-9
 - page-cleaner table 12-28
 - parameters 4-11
 - performance considerations 12-5
 - performance options 13-14
 - physical-log buffer 12-23
 - pools 12-25
 - portions 12-7
 - purposes of 12-5
 - reinitializing 13-16
 - resident portion 13-9
 - resident portion, mentioned 9-7
 - segment identifier 12-14
 - semaphore guidelines 13-7
 - session control block 12-30
 - session data 12-30
 - SHMADD parameter 12-25
 - SHMTOTAL parameter 4-10, 12-9
 - SHMVIRTSIZE parameter 4-10, 12-25
 - size of virtual portion 12-25
 - size, displayed by onstat 12-9
 - smart large objects 14-34
 - sorting 12-32
 - SQL statement cache 12-8, 12-32
 - stacks 12-31
 - STACKSIZE parameter 12-31
 - synchronizing buffer flushing 12-46
 - tables 12-26
 - tblspace table 12-29
 - the resident portion 12-18
 - thread control block 12-30
 - thread data 12-30
 - thread isolation and buffer locks 12-35
 - transaction table 12-29
 - user table 12-30
 - user-defined routine cache 12-33
 - virtual portion 12-25, 12-26, 13-12, 13-18
- Shared-memory connection example 5-50
 - how a client attaches 12-11
 - in servicename field 5-33
 - message 10-35
 - message buffers 12-33
 - virtual processor 10-32
- SHMADD parameter 4-10, 12-25
- SHMBASE parameter
 - attaching first shared-memory segment 12-12, 12-13
 - description of 4-10, 12-14
 - setting higher 13-6
 - warning 12-14
- shmenv file
 - and assertion failures 29-7
- shmkey
 - attaching additional segments 12-15
 - description of 12-13
- SHMTOTAL parameter 4-10, 12-9
- SHMVIRTSIZE parameter 4-10, 12-25
- Shutdown
 - graceful 8-7
 - immediate 8-8
 - mode, description of 8-4
 - taking off-line 8-9
- Shutdown script
 - multiple residency 7-12
 - steps to perform 3-20
- Simple large object
 - and blobspaces 1-22
 - creating in a blobspace 12-57
 - description of 1-22
 - illustration of blobspace storage 12-57
 - See also* TEXT and BYTE data.
- Simple large objects
 - buffers 12-56
 - descriptor 12-56
 - writing to disk 12-56
- Single processor computer 10-21
- SINGLE_CPU_VP parameter
 - and single processor computer 10-21, 10-22
- Sizing guidelines

- logical log 19-7
- physical log 21-6, 21-7
- Skipping fragments
 - all fragments 15-42
 - all unavailable 15-38, 15-42
 - effect on transactions 15-41
 - selected fragments 15-42
 - when to use feature 15-41
- Sleep queues, description of 10-17
- Sleeping threads
 - forever 10-18
 - types of 10-17
- Smart large object 14-14
 - accessing in applications 14-34
 - and buffers 12-58
 - and dbspaces 14-15
 - and sbpages 14-14
 - and sbspaces 1-23
 - API 1-23
 - buffering mode 14-32
 - buffering recommendation 14-32
 - byte-range locking 12-24
 - calculating size 14-30
 - creating 14-27
 - data retrieval 14-25
 - deleting CLOB or BLOB
 - data 15-31, 15-33
 - description of 1-22, 14-24
 - estimated average size 15-27
 - extent size 14-29, 14-30
 - fragmentation 16-7
 - I/O properties 14-25
 - last-access time 14-32
 - lightweight I/O 12-20, 14-32
 - locking 14-25, 14-33
 - logging 12-58, 14-33
 - memory 14-34
 - metadata 14-26
 - physical logging 21-5
 - shared-memory buffer 12-58
 - sizing metadata area 15-25
 - specifying data types 14-27
 - statistics for user-defined
 - data 4-5
 - storage characteristics 14-29, 14-34, 15-24, 15-28
 - user data 14-26
- Smart-large-object extent,
 - relationship to chunk 14-15

- SMI table
 - aborted table build 9-11
 - during initialization 9-11
 - monitoring buffer use 13-24
 - monitoring buffer-pool 13-27
 - monitoring checkpoints 23-26
 - monitoring chunks 15-52
 - monitoring data replication 28-26
 - monitoring databases 15-43, 18-11
 - monitoring fragmentation 15-56
 - monitoring latches 13-29
 - monitoring log buffer use 22-9
 - monitoring logical-log files 20-19
 - monitoring shared memory 13-21
 - monitoring virtual
 - processors 11-15
 - preparation during
 - initialization 9-11
 - See also Administrator's Reference;*
 - System-monitoring interface.*
- Sockets
 - in nettype field 5-30
- Software dependencies Intro-4
- Sorting
 - and shared memory 12-32
 - as parallel process 10-10
- SPL routine
 - PC_HASHSIZE parameter 4-11
 - PC_POOLSIZE parameter 4-11
 - UDR cache hash size 4-11
 - UDR cache size 4-11
 - user-defined routine 1-24
- SPL routine cache
 - specifying hash size 4-11, 13-13
 - specifying size 4-11, 13-13
- SPL routine cache. *See* UDR
 - cache. 12-33
- Split read 24-10
- SQL code Intro-11
- SQL operator
 - description of 16-14
 - scan 16-14
- SQL statement cache
 - description 12-32
 - shared-memory location 12-8
 - specifying size 4-12, 13-14
 - specifying use 4-12, 13-14
- SQL statements

- ALTER FRAGMENT 12-21
- CREATE DBSPACE 14-20
- CREATE TABLE 14-20
- CREATE TABLE,PUT
 - clause 14-27
- SET INDEX 12-21
- SET TABLE 12-21
 - using temporary disk space 14-39
- sqlexec thread
 - and client application 10-13
 - as user thread 10-5
 - role in client/server
 - connection 10-35
- sqlhosts
 - and multiplexed connection 5-7
 - CSM option 5-36
 - dbservername field 5-28
 - keep alive option 5-39
 - local-loopback example 5-51
 - multiple connection types,
 - example 5-48
 - multiple residency 7-10
 - network connection example 5-52
 - security options 5-40
 - shared-memory example 5-50
- sqlhosts file 3-14
 - communication support module
 - option 5-36
 - defining multiple network
 - addresses 10-37
 - description of 5-23
 - end of group option 5-36
 - entries for multiple interface
 - cards 10-38
 - for initialization 3-17
 - group option 5-37
 - identifier option 5-39
 - mentioned 1-8
 - specifying network poll
 - threads 10-32
- sqlhosts registry
 - central registry 5-24
 - description of 5-24
 - INFORMIXSQLHOSTS
 - environment variable 5-25
 - location of 5-24
- Stack
 - and thread control block 10-16
 - description of 10-15

INFORMIXSTACKSIZE
 environment variable 12-31
 pointer 10-16
 size of 12-31
 STACKSIZE parameter 12-31
 thread 12-31
STACKSIZE parameter
 changing the stack size 12-31
 description 12-31
Standard database server 27-5
Starting the database server
 and initializing disk space 3-17
Startup script
 multiple residency 7-12
 multiple versions of the database
 server 3-20
Statistics. *See* onstat utility.
STMT_CACHE configuration
 parameter 4-12, 13-14
STMT_CACHE_SIZE 4-12, 13-14
Storage characteristics
 hierarchy 14-31
 sbspaces 14-29
 smart large objects 14-34, 15-24,
 15-28
Storage devices, setup 3-22
Storage manager
 ISM 3-22
 role in ON-Bar system 3-22
Storage statistics
 blobpages 15-34
 blobspaces 15-34
Stored procedure. *See* SPL routine.
Stored-procedure cache. *See* UDR
 cache. 12-33
stores_demo database Intro-5
Stream-pipe connection
 advantages and
 disadvantages 5-12
 in servicename field 5-33
Structured Query Language 1-6
 INSERT statement 16-7
 See also SQL statement.
superstores_demo database Intro-5
Swapping memory 12-17
Switching between threads 10-16
Symmetric multiprocessing,
 description of 1-8
Sync checkpoint. *See* Checkpoint.

sysmaster database
 See SMI table; System-monitoring
 interface.
System catalog tables
 and dictionary cache 12-32
 location of 14-36
 optimal storage of 14-51
 sysfragments table 16-4
System failure, defined 23-15
System requirements
 database Intro-4
 software Intro-4
System startup script, multiple
 residency 7-12
System timer 10-17
System-defined hash distribution
 scheme
 definition 16-21, 16-23
System-monitoring interface (SMI)
 See also SMI table.
 using to monitor database
 server 2-15

T

Table
 access methods 1-25
 creating temporary 16-9
 creating with CLOB or BLOB data
 types 14-34
 description of 14-37
 disk-layout guidelines 14-52
 high-use 14-55
 inheritance 1-20
 isolating 14-52
 migration. *See Informix Migration
 Guide.*
 recommendations for
 storage 14-55
 relationship to extent 14-37
 storage considerations 14-37
 storage on middle partition of
 disk 14-55, 14-59
 temporary
 cleanup during shared-memory
 initialization 14-39
 estimating disk space for 14-46
 storage of explicit 14-40
 storage of implicit 14-41
Tbospace
 description of 14-19, 14-42
 temporary tbospace during
 initialization 9-10
 types of pages contained in 14-42
Tbospace table
 contents of 12-29
 description of 12-29
TCP/IP communication protocol
 in hostname field 5-31
 in servicename field 5-32
 listen port number 5-46
 using 5-17, 5-18
 using a wildcard 5-43
 using the internet IP address 5-42
 using the TCP listen port
 number 5-46
TEMP TABLE clause, of CREATE
 TABLE 16-9
Temporary dbspace
 and data replication 27-35, 28-7
 and DBSPACETEMP 14-40
 creating 15-16
 description 14-22
Temporary disk space
 amount required for temporary
 tables 14-46
 operations requiring 14-39
Temporary table
 and fragmentation 16-9
 during initialization 9-10
 explicit 14-38
 where stored 14-40
Testing environment 6-4
TEXT and BYTE data
 descriptor 23-13
 descriptor, time stamp associated
 with 23-13
 effect of Committed Read
 isolation 23-14
 effect of Dirty Read
 isolation 23-14
 entering 15-64
 monitoring in a dbspace 15-61
 role of time stamps 23-13
 writing to a blobspace 12-55
See also Simple large object.

- TEXT data type
 - mentioned 1-22
 - See TEXT and BYTE data.
 - Text editor
 - setting performance configuration parameters 13-15
 - setting shared-memory parameters 13-11, 13-13
 - setting virtual-processor parameters 11-4
 - Thread
 - access to resources 10-10
 - accessing shared buffers 12-36
 - and heaps 12-32
 - and stacks 12-31
 - concurrency control 12-34
 - context of 10-13
 - control block 10-13, 12-30
 - description of 10-4
 - dynamic allocation of 1-12
 - for client applications 10-3
 - for primary session 10-13
 - for recovery 10-5
 - how virtual processors service 10-12
 - internal 10-5, 10-19
 - kernel asynchronous I/O 10-30
 - main_loop() 23-10
 - migrating 10-16
 - mirroring 10-5
 - multiple concurrent 10-13
 - ON-Monitor 10-5
 - page cleaner 10-5, 12-45
 - relationship to a process 1-9, 10-4
 - scheduling and synchronizing 10-13
 - session 10-5, 10-19
 - sleeping 10-18, 12-44
 - supporting data replication 27-14
 - switching between 10-16
 - user 10-5
 - waking up 10-17
 - yielding 10-13
 - Thread control block
 - creation of 12-30
 - role in context switching 10-14
 - Thread-safe virtual processor 10-7
 - Time stamp
 - associated with TEXT and BYTE data 23-13
 - blobpage pair 23-13
 - description of 23-12
 - mentioned 12-45
 - page-header and page-ending pair 23-12
 - role in
 - data consistency 23-12
 - flushing physical-log buffer 12-49
 - Tip icons Intro-10
 - TLL. *See* Transport-layer interface.
 - Transaction
 - factors which prevent closure 19-20
 - global
 - definition of 30-5
 - determining if implemented consistently 31-4
 - identification number, GTRID 31-8
 - tracking 30-28
 - mixed result 30-25
 - piece of work, definition of 30-5
 - two-phase commit, examples 30-8
 - Transaction logging
 - buffered 17-13
 - definition of 17-6
 - unbuffered 17-12
 - when to use 17-12
 - See also Logging.
 - Transaction table
 - description of 12-29
 - tracking with onstat 12-29
 - Transport-layer interface
 - in nettype field 5-30
 - Trusted clients, database server and Windows NT domains 5-6
 - Trusted domain 5-6
 - Trusted domain, Windows NT 5-6
 - Two-phase commit
 - contrast with heterogeneous commit 30-38
 - Two-phase commit protocol
 - automatic recovery 30-10
 - administrator's role 30-10
 - mechanisms for coordinator recovery 30-10
 - mechanisms for participant recovery 30-14
 - race condition 30-17
 - configuration parameters for 30-37
 - coordinator definition 30-5
 - coordinator recovery mechanism 30-11
 - description of 30-3, 30-4
 - errors messages for 30-29
 - global transaction definition 30-5
 - global transaction identification number 31-8
 - heuristic decisions
 - heuristic end-transaction 30-26
 - heuristic rollback 30-22
 - types of 30-21
 - independent action
 - definition of 30-18
 - resulting in error condition 30-20
 - resulting in heuristic decisions 30-21
 - results of 30-19
 - what initiates 30-19
 - logical-log records for 30-30
 - messages 30-6
 - participant recovery 30-14
 - participant, definition of 30-5
 - piece of work, definition of 30-5
 - post-decision phase 30-6, 30-9
 - precommit phase 30-6
 - presumed-abort
 - optimization 30-10, 30-17
 - requirements for flushing logical log records 30-32
 - role of current server 30-5
 - use of DEADLOCK_TIMEOUT 30-37
 - use of TXTIMEOUT 30-37
- TXTIMEOUT parameter and onmode -Z 30-27
- description of 30-37
 - in two-phase commit protocol 30-37
 - role in automatic recovery 30-11, 30-15, 30-17

Types of applications 16-16
Types of buffer writes 12-50

U

UDR cache 12-33
Unbuffered disk access
 compared to buffered 14-7
 raw disk devices 1-11
Unbuffered disk space
 in data storage 14-6
Unbuffered logging
 flushing the logical-log
 buffer 12-54
Unbuffered transaction logging. *See*
 Logging.
Unique ID 19-11
Units of storage 14-3
UNIX devices
 creating a link to a
 pathname 15-11
 displaying links to a
 pathname 3-8, 15-12
 ownership, permissions on
 character-special 15-11
UNIX files
 ownership, permissions on
 cooked files 15-9
UNIX operating system
 default locale for Intro-4
 link command 3-8, 15-11
 shutdown script 3-20
Upgrading the database
 server 3-10
User accounts and Windows NT
 domains 5-5
User data
 creating 15-26
 definition 14-26
 estimating size of 15-24
User impersonation 5-20
User table
 description of 12-30
 maximum number of
 entries 12-30
User thread
 acquiring a buffer 12-42
 description of 10-5

 in critical sections 23-4
 monitoring 12-30
 tracking 12-30
User-data area, of sbspace 12-58
User-defined access methods
 mentioned 1-18
User-defined data types
 mentioned 1-18, 1-20, 1-21
User-defined routines
 configuring cache 12-33
 description of 1-24
 ill-behaved 10-25
 memory cache 12-33
 mentioned 1-18, 1-24
 shared-memory location 12-8,
 12-34
 virtual processors for 10-21
User-defined routine, parallel
 processing of 10-11
User-defined virtual processors
 mentioned 1-18
 purpose of 10-25
 running UDRs 12-34
 when to use 10-25
Users, types of Intro-3
Utility
 Find Error Intro-13
Utilities
 onmode
 -a option 13-18
 adding a segment 13-18
 onstat
 -g seg option 13-6
Utility
 attaching to shared-
 memory 12-12
 oncheck
 -pr option 21-6
 onperf, description of 1-17
 rofferr Intro-13
 See also Administrator's Reference.

V

Virtual extension portion (shared
 memory) 12-7
 operating-system segments,
 and 12-7

Virtual portion (shared memory)
 adding a segment 13-18
 configuration 12-25, 13-12
 contents of 12-25, 12-26
 global pool 12-33
 SHMVIRTSIZE parameter 4-10,
 12-25
 SQL statement cache 12-32
 stacks 12-31
 UDR cache 12-33
Virtual processing
 sharing processing 10-9
Virtual processor
 access to shared memory 12-7
 adding and dropping 10-11
 ADM class 10-17
 ADT class 10-39
 advantages 10-9
 AIO class 10-30
 AIO, how many 10-31
 and context switching 10-10
 and ready queue 10-17
 as multithreaded process 1-9,
 10-4
 attaching to shared memory 12-7,
 12-12
 binding to CPUs 10-12
 classes of 10-6, 10-19
 coordination of access to
 resources 10-10
 CPU class 10-19
 description of 10-3
 disk I/O 10-27
 dropping (CPU) in on-line
 mode 11-11
 during initialization 9-8
 extension. *See* User-defined
 virtual processor
 for DataBlade modules 10-21
 for user-defined routine 10-21
 how threads serviced 10-12
 LIO class 10-27
 LIO, how many 10-29
 logical-log I/O 10-28
 monitoring 11-12
 moving a thread 10-9
 MSC (miscellaneous) class 10-40
 network 10-32, 10-33
 OPT (optical) class 10-39

parallel processing 10-10
 physical log I/O 10-29
 PIO class 10-27
 PIO, how many 10-29
 setting configuration
 parameters 11-3
 sharing processing 10-9
 UDR written in Java 10-27
 use of stack 10-15
 user-defined 10-24
 class 10-7
 VP class in NETTYPE
 parameter 10-33
 VPCLASS parameter
 configuring virtual
 processors 10-20
 mentioned 10-7
 set with ON-Monitor 11-5

W

Wait queue
 and buffer locks 12-36
 description of 10-18
 Waking up threads 10-17
 Warning
 files on NIS systems 5-16
 interpreting after running
 oncheck -cc 29-6
 Warning icons Intro-10
 Wildcard addressing
 by a client application 5-45
 example 5-44
 in hostname field 5-43
 Windows Internet Name Service,
 mentioned 5-17
 Windows NT
 allocating raw disk space 15-12
 client connections 26-5
 default locale for Intro-4
 Event Viewer 2-15
 high availability 26-3
 larger shared memory 13-5
 shared memory issues 13-6
 virtual memory 13-5
 WORM. *See* Optical devices.

Write types
 chunk write 12-52
 foreground write 12-51
 LRU write 12-51

X

X/Open compliance level Intro-15

Y

Year 2000 compliance 1-17
 Year values 1-17
 Yielding threads
 and ready queue 10-17
 at predetermined point 10-14
 description of 10-13
 on some condition 10-14
 switching between 10-12
 ypcat hosts command 5-16
 ypcat services command 5-16

Symbols

.informix file 3-13
 .netrc file 5-19
 sqlhosts security options 5-40
 .rhosts file 5-19

