

Oracle[®]

Performance Tuning and Optimization

Edward Whalen



*201 West 103rd Street
Indianapolis, Indiana 46290*

To my father.

Copyright 1996 by Sams Publishing

FIRST EDITION

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein. For information, address Sams Publishing, 201 W. 103rd St., Indianapolis, IN 46290.

International Standard Book Number: 0-672-30866-X

Library of Congress Catalog Card Number: 95-72345

99 98 97 96 4 3 2 1

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit, the number of the book's printing. For example, a printing code of 96-1 shows that the first printing of the book occurred in 1996.

Composed in AGaramond and MCPdigital by Macmillan Computer Publishing

Printed in the United States of America

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Publisher and President: *Richard K. Swadley*

Acquisitions Manager: *Greg Wiegand*

Development Manager: *Dean Miller*

Managing Editor: *Cindy Morrow*

Marketing Manager: *Gregg Bushyager*

Acquisitions Editor

Rosemarie Graham

Development Editors

Byron Pearce

Todd Bumbalough

Software Development

Specialist

Steve Flatt

Production Editor

Alice Martina Smith

Technical Reviewers

David Kennedy

Stephen Tallon

Editorial Coordinator

Bill Whitmer

Technical Edit Coordinator

Lynette Quinn

Formatter

Frank Sinclair

Editorial Assistants

Sharon Cox

Andi Richter

Rhonda Tinch-Mize

Cover Designer

Tim Amrhein

Book Designer

Alyssa Yesh

Copy Writer

Peter Fuller

Production Team Supervisor

Brad Chinn

Production

Mary Ann Abramson

Georgianna Briggs

Jama Carter

Amy Chinn

Michael Dietsch

Jason Hand

Sonja Hart

Ayanna Lacey

Clint Lahnen

Paula Lowell

Brian-Kent Proffitt

Bobbi Satterfield

Susan Van Ness

Colleen Williams

Overview

Introduction	xxiv
PART I Introduction 1	
1 Introduction to Oracle	3
2 Understanding Terms	21
3 What Is a Well-Tuned System?	31
4 Tuning Methodology	41
5 Benchmarking	51
6 Performance Monitoring Tools.....	73
7 Performance Engineering Starts at the Design Stage	81
PART II Tuning the Server 89	
8 What Affects Oracle Server Performance?	91
9 Oracle Instance Tuning.....	97
10 Performance Enhancements	139
11 Tuning the Server Operating System.....	167
12 Operating System-Specific Tuning	177
13 System Processors	205
14 Advanced Disk I/O Concepts	213
15 Disk Arrays	225
PART III Configuring the System 243	
16 OLTP System	245
17 Batch Processing System.....	265
18 Decision Support System	285
19 Data Warehousing System.....	303
20 BLOB System	323
21 The Oracle Parallel Server System	339
22 Optimal Backup and Recovery	349
23 Miscellaneous Configurations	367
PART IV Tuning SQL 391	
24 What Is a Well-Tuned SQL Statement?	393
25 Using EXPLAIN PLAN and SQL Trace	403
26 Tuning SQL Statements.....	419
27 Using the Oracle Optimizer	437
28 Using Procedures, Functions, and Packages	449
29 Providing for Data Integrity and Triggers.....	461
30 Using Hints	475
31 Introducing SQL Development Tools	489
32 Miscellaneous SQL Topics	501

PART V Tuning the Client 513	
33 What Affects Client Performance?	515
34 Tuning the Client System	525
35 Using GUI Builders	533
36 Using Middleware Products	555
PART VI Tuning the Network 563	
37 What Affects Network Performance?	565
38 Tuning the Network Components	573
PART VII References 579	
A Review of Tuning Guidelines	581
B Quick Reference	595
C Flowcharts	603
D Glossary	607
E Oracle Tuning Parameters	619
F Contents of the CD-ROM	645
Index	649

Contents

Introduction	xxiv
Part I Introduction 1	
1 Introduction to Oracle	3
The Database	4
The Physical Layer	4
The Logical Layer	5
The Oracle Instance	8
The Oracle Memory Structure	8
System Global Area (SGA)	9
Program Global Area (PGA)	10
Processes	10
How Transactions Work	12
Oracle Products	13
Oracle RDBMS Products	13
Oracle Workgroup Server	15
Personal Oracle for Windows	16
Oracle Development Tools	16
Oracle Applications	17
Oracle Services	18
Summary	19
2 Understanding Terms	21
Terms	22
RDBMS Functionality	26
Checkpoint	26
Logging and Archiving	26
Business Models	27
OnLine Transaction Processing (OLTP)	27
Batch Processing	27
Decision Support	28
Data Warehousing	28
Binary Large Objects (BLOBs)	28
Unit Conversions	28
Powers of 10	29
Storage Units	29
Summary	30
3 What Is a Well-Tuned System?	31
Client/Server Computing	32
The Client or Front-End Machines	33
The Server	33
The Network	35
Client/Server Checklist	35

Host-Based Computing.....	36
The Front-End Application.....	36
The Database	36
Terminal-Based Checklist.....	37
Batch Computing	38
Batch-Processing Checklist.....	39
Exceptions	39
Multimedia Systems	39
Shipping Systems	39
Summary.....	40
4 Tuning Methodology	41
Goals	42
Throughput	42
Response Time.....	42
Connectivity	43
Fault Tolerance	43
Load Time	44
Tuning Methodology	44
Examine the Problem	45
Determine the Problem	47
Determine the Solution and Set Goals	48
Test the Solution	49
Analyze the Results	50
Summary.....	50
5 Benchmarking	51
Introduction to Benchmarking	52
Industry Standard Benchmarks	52
The Transaction Processing Performance Council (TPC)	53
TPC Rules and Regulations	55
Results	56
Benchmarks	57
Publication Benchmarks	69
Custom Benchmarks	70
Writing Your Own Benchmark	70
Summary.....	72
6 Performance Monitoring Tools	73
Oracle Tools.....	75
SQL*DBA Monitor	76
Server Manager	76
Oracle SNMP Agents	76
SQL Trace	76
EXPLAIN PLAN	77

OS Tools	77
Third-Party Tools	78
Real-Time Monitors	79
Threshold Monitors	79
Summary.....	80
7 Performance Engineering Starts at the Design Stage	81
Design Stage	82
Database Layout	82
Indexes and Clusters	83
Application Design	83
Hardware Sizing	83
Network Considerations	84
Performance Tuning after the System Is Built	84
Tuning the Client	85
Tuning the Server	85
Tuning the Network.....	85
Summary.....	86
Part II Tuning the Server 89	
8 What Affects Oracle Server Performance?	91
System Bottlenecks	92
Finding the Bottleneck	92
Removing the Bottleneck	93
System Tuning	93
Tuning RDBMS Resources	93
Tuning OS Resources	94
Tuning Hardware Resources	94
Other Tuning Factors	95
System Limitations	95
Summary.....	95
9 Oracle Instance Tuning.....	97
Tuning Memory	98
Tuning the Operating System	99
Tuning the Private SQL and PL/SQL Areas	100
Tuning the Shared Pool	100
Tuning the Buffer Cache	107
Tuning the I/O Subsystem	108
Understanding Disk Contention	109
Identifying Disk Contention Problems	110
Solving Disk Contention Problems.....	111
Reducing Unnecessary I/O Overhead.....	117
Migrated and Chained Rows	117
Dynamic Extensions	118

PCTFREE and PCTUSED Command Options	119
A Review of I/O Reduction Techniques.....	122
Tuning Rollback Segments	123
Understanding How Rollback Segments Work	123
Tuning Rollback Segments	126
Review of Rollback Segment Tuning	130
Checking for Latch Contention	130
Redo Log Buffer Contention	130
Redo Log Buffer Latch Contention	131
Tuning Checkpoints	133
Optimizing Archiving	134
Adjusting the Effect of Archiving.....	135
Optimizing Sorts	135
Minimizing Free List Contention	136
Summary.....	137
10 Performance Enhancements	139
Block Size	140
Clusters	141
Direct-Write Sorts	143
Fragmentation	144
Hash Clusters	146
When To Hash	147
Indexes	148
Index Types	149
How the Oracle Index Works	149
What To Index	150
Multiblock Reads	152
Multiblock Writes	152
Parallel Query Option	153
Parallel Query Processing	153
Direct-Write Sorts	158
Parallel Index Creation	159
Parallel Loading.....	159
Parallel Recovery	160
Parallel Server Option	161
Spin Counts	164
Summary.....	164
11 Tuning the Server Operating System.....	167
Goals	168
Processes	169
Memory	170
I/O	170

Direct or Synchronous I/O	172
Asynchronous I/O	172
Miscellaneous	173
Post-Wait Semaphore	173
Scheduling and Preemption	173
Cache Affinity	174
Summary.....	174
12 Operating System-Specific Tuning	177
NetWare	178
Architectural Overview	178
Tuning Considerations	179
Windows NT	183
Architectural Overview	183
Tuning Considerations	185
OS/2	188
Architectural Overview	188
Tuning Considerations	188
UNIX	191
Architectural Overview	192
Tuning Considerations	192
Summary.....	203
13 System Processors	205
Overview of Computer Architecture	206
CPU and Cache	206
CPU Design	207
CISC Processors	207
RISC Processors	208
Multiprocessor Systems	209
SMP Systems	209
MPP Systems	209
CPU Cache	210
System Memory Architecture	210
Virtual Memory System	211
Bus Design	211
Summary.....	212
14 Advanced Disk I/O Concepts	213
Disk Operation	214
Seek Time	216
Rotational Latency	217
Data Transfer Rate.....	217
Queue Time	218

Disk Performance	219
Random I/Os	220
Sequential I/Os	220
Summary.....	223
15 Disk Arrays	225
How Does a Disk Array Work?	226
Software Array	227
Hardware Array.....	227
RAID Technology	229
RAID-0	230
RAID-1	230
RAID-2	231
RAID-3	231
RAID-4	232
RAID-5	232
Fault-Tolerance Concerns	233
No Data Protection	233
Full Data Protection	234
Partial Data Protection	234
Configuring RAID for RDBMS Performance	235
Isolate Sequential I/Os	236
Distribute Random I/Os	237
Size the Volume Properly	238
Configure for the Disk Array.....	240
RAID Comparison	240
Summary.....	241
Part III Configuring the System 243	
16 OLTP System	245
Characteristics of the OLTP System.....	246
Data Access Patterns	246
System Load.....	247
Goals	248
Design Considerations	249
Physical Data Layout	250
Hardware Considerations.....	253
Tuning Considerations	253
Oracle Tuning.....	254
Server OS Tuning	255
Enhancements	256
Oracle Parallel Server Option	257
Hardware Enhancements	257

Performance Verification	260
What To Test in the RDBMS	261
What To Test in the OS	261
Benchmarks	262
Summary.....	262
17 Batch Processing System	265
Characteristics of the Batch Processing System	266
Data Access Patterns	267
System Load	267
Goals	268
Design Considerations	270
Physical Data Layout	270
Hardware Considerations.....	274
Tuning Considerations	274
Oracle Tuning.....	274
Server OS Tuning	276
Enhancements	277
Parallel Query Option	277
Oracle Parallel Server Option	278
Hardware Enhancements	279
Performance Verification	281
What To Test in the RDBMS	281
What To Test in the OS	282
Benchmarks	282
Summary.....	283
18 Decision Support System	285
Characteristics of a DSS System	287
Data Access Patterns	287
System Load	288
Goals	289
Design Considerations	290
Physical Data Layout	291
Hardware Considerations.....	294
Tuning Considerations	294
Oracle Tuning.....	294
Server OS Tuning	295
Enhancements	296
Parallel Query Option	297
Oracle Parallel Server Option	297
Hardware Enhancements	298
Performance Verification	300

What To Test in the RDBMS	301
What To Test in the OS	301
Benchmarks	301
Summary.....	302
19 Data Warehousing System	303
Characteristics of a Data Warehouse	304
Data Access Patterns	305
System Load.....	306
Goals	307
Design Considerations	308
Physical Data Layout	308
Fault Tolerance Consideration	311
Hardware Considerations.....	312
Tuning Considerations	312
Oracle Tuning.....	312
Server OS Tuning	314
Enhancements	315
Parallel Query Option	316
Oracle Parallel Server	316
Hardware Enhancements	317
Performance Verification	319
What To Test in the RDBMS	320
What To Test in the OS	320
Benchmarks	320
Summary.....	321
20 BLOB System	323
Characteristics of BLOBs	324
Data Access Patterns	324
System Load.....	324
Goals	325
Design Considerations	327
Physical Data Layout	328
Hardware Considerations.....	331
Tuning Considerations	331
Oracle Tuning.....	332
Server OS Tuning	333
Enhancements	333
Hardware Enhancements	334
Performance Verification	335
What To Test in the RDBMS	336
What To Test in the OS	336
Benchmarks	336
Summary.....	337

21	The Oracle Parallel Server System	339
	Oracle Parallel Server Architecture	340
	Design Considerations	343
	Design Goals	343
	System Design	346
	Tuning the Parallel Server System.....	346
	Summary.....	348
22	Optimal Backup and Recovery.....	349
	RDBMS Operational Review	351
	Backup Process	351
	Recovery Process	351
	Characteristics of the Oracle Backup Process	352
	Cold (Offline) Backup	352
	Hot (Online) Backup	352
	Data Access Patterns During Backup	353
	System Load During Backup	353
	Backup Goals	353
	System Design Considerations	354
	Cold Database Backup	354
	Hot Database Backup	355
	Tuning Considerations	358
	System Enhancements To Improve Backup Performance	359
	CPU Enhancements	359
	I/O Enhancements	359
	Network Enhancements	360
	Split Up the Backup	360
	Performance Verification	362
	What To Test in the RDBMS	362
	What To Test in the OS	363
	Summary.....	365
23	Miscellaneous Configurations	367
	Financial Systems	368
	System Characteristics	369
	Design and Tuning Hints	369
	Enhancements	372
	Replicated Systems	374
	System Characteristics	374
	Design and Tuning Hints	375
	Distributed Systems	377
	System Characteristics	378
	Design and Tuning Hints	378

TextServer 3.0 Systems.....	379
System Characteristics	379
Design and Tuning Hints	380
Enhancements	381
Oracle Office Systems	382
System Characteristics	383
Design and Tuning Hints	383
WebServer Systems	386
System Characteristics	386
Design and Tuning Hints	387
Enhancements	389
Summary.....	390
Part IV Tuning SQL 391	
24 What Is a Well-Tuned SQL Statement?	393
How To Identify Badly Formed SQL Statements	394
Transaction Processing	395
SQL Statement Processing.....	397
Cursor Creation	398
Statement Parsing.....	399
Query Processing.....	400
Bind Variables	401
Statement Execution	401
Parallelization	401
Fetch Rows To Be Returned	401
Summary.....	402
25 Using EXPLAIN PLAN and SQL Trace	403
SQL Trace	404
SQL Trace Initialization	404
Controlling SQL Trace	405
SQL Trace Functionality	406
TKPROF Functionality	407
Interpreting SQL Trace	409
The EXPLAIN PLAN Command	414
EXPLAIN PLAN Initialization	414
Invoking EXPLAIN PLAN	415
Extracting EXPLAIN PLAN Results	416
Registering Applications	417
Summary.....	418
26 Tuning SQL Statements	419
Tuning an Existing Application	420
Problem Analysis	420
Tuning the Application	422

Designing a New Application	426
Indexes	426
Clusters	430
Hash Clusters	431
Packages, Procedures, and Functions	432
Optimization Approaches	433
Discrete Transactions	435
Summary	436
27 Using the Oracle Optimizer	437
How the Optimizer Works	438
How To Specify an Optimization Mode	438
Optimization Methods	439
Rule-Based Approach	440
Cost-Based Approach	441
Using the ANALYZE Command	441
How To Run the ANALYZE Command	442
Data Dictionary Statistics	445
Hints	447
Summary	447
28 Using Procedures, Functions, and Packages	449
Review of the Library Cache	450
Procedures and Functions	452
Procedures	453
Functions	453
How Procedures and Functions Operate	454
How To Create Stored Procedures and Stored Functions	456
How To Replace Procedures and Functions	457
Packages	457
Summary	459
29 Providing for Data Integrity and Triggers	461
Integrity Constraints	462
Referential Integrity	462
Integrity Constraints	465
Using Constraints	466
Triggers	469
Using Triggers	469
Using Alerts	470
Creating Triggers	470
Viewing Triggers	471
Audit Trails	472
Serial Reads	473
Summary	473

30	Using Hints	475
	Implementing Hints	476
	Hint Syntax	477
	Hint Errors	477
	Using Multiple Hints	478
	Hints	478
	Optimization Approaches	478
	Access Methods	481
	Parallel Query Hints	485
	Summary.....	487
31	Introducing SQL Development Tools	489
	Database Design Tools	490
	Oracle Designer/2000	490
	Third-Party Tools	492
	Application Development Tools	494
	Oracle Tools	494
	Third-Party Tools	495
	Analysis Tools	496
	Oracle Mission Control	496
	Third-Party Tools	497
	Summary.....	499
32	Miscellaneous SQL Topics	501
	Table Sequences	502
	Creating Sequences	502
	Tuning Sequences	503
	Using Sequences	503
	Using Cached Sequences for Primary Key Values	504
	Join Performance	505
	Equijoin	506
	Self Join	506
	Cartesian Product	506
	Outer Join	507
	Tuning Joins for Throughput	507
	Tuning Joins for Response Time	507
	Locking	508
	What Is Locking?	508
	Serializable Reads	508
	Using Locks	509
	Array Processing	510
	Using VARCHAR2 instead of CHAR	510
	Summary.....	511

Part V Tuning the Client 513

33 What Affects Client Performance?	515
What Is a Client Machine?	516
The Traditional Computing Model	516
The Network Computing Model	517
The GUI/Server Model.....	517
The Client/Server Model	519
Two-Tiered and Three-Tiered Models	520
Two-Tiered System	520
Three-Tiered System	521
Client Bottlenecks.....	522
Network Performance	523
Application Performance.....	523
Presentation Performance	524
Client Hardware Performance	524
Summary.....	524
34 Tuning the Client System.....	525
Windows NT	527
Tuning Memory.....	527
16-bit Applications	527
I/O Performance	528
Microsoft Windows 3.1 and Windows for Workgroups 3.11	528
Memory	528
Network	528
Microsoft Windows 95	529
32-Bit Support	529
Memory	530
Network	530
Oracle Support	530
UNIX	530
Memory	531
Network	531
Hardware	531
Summary.....	532
35 Using GUI Builders	533
Tuning the Application	534
First-Generation Graphical Application Development Tools	534
Modern Graphical Application Development Tools	535
How To Test and Improve Automatically Generated SQL Statements	535
Oracle Tools	536
Developer/2000	536
Power Objects	544

Third-Party Tools	546
Delphi from Borland	547
ReportSmith	548
SQLWindows from Gupta.....	550
PowerBuilder from Powersoft.....	552
Summary.....	553
36 Using Middleware Products	555
What Is Middleware?	556
Two-Tiered System Architecture	556
Three-Tiered System Architecture	557
Application Servers	558
How To Tune the Application Server	559
Transaction Monitor (TM)	559
What Is a TM?	559
When To Use a Transaction Monitor	561
Tuning the TM and System	561
Summary.....	562
Part VI Tuning the Network 563	
37 What Affects Network Performance?	565
Network Architecture	566
Hardware Components.....	566
Summary.....	571
38 Tuning the Network Components	573
Software Tuning	574
NetWare	574
Windows NT	575
OS/2	575
UNIX	575
Oracle Tuning.....	575
Network Design	576
Bandwidth Considerations	576
Segmenting the Network	577
Bridges, Routers, and Hubs.....	577
Summary.....	578
Part VII References 579	
A Review of Tuning Guidelines	581
RDBMS Tuning.....	582
SGA	582
Performance Enhancements	583
OS Tuning.....	588
OS Tuning Goals	589
OS Features	589

I/O Tuning	590
System Design	590
Application Tuning	591
Client Tuning	592
Network Tuning.....	593
B Quick Reference	595
Oracle Instance Tuning.....	596
Library Cache.....	596
Data Dictionary Cache	596
Database Buffer Cache	597
Physical I/O Usage	597
Chained Rows	599
Recursive Calls	599
Rollback Segment Contention.....	599
Dynamic Rollback Growth	600
Redo Log Buffer Contention.....	600
Redo Latch Contention	601
Sort Performance	601
Free List Contention	602
C Flowcharts	603
Problem-Solving Methodology	604
User-Transaction Profile	605
SQL Statement Processing.....	605
The Oracle Optimizer	605
D Glossary	607
E Oracle Tuning Parameters	619
Performance Parameters	620
Parallel Query Option Parameters.....	626
Analysis Tool Parameters.....	627
General Parameters	629
Multithreaded Server Parameters	637
Distributed Option Parameters.....	638
Parallel Server Parameters	639
Security Parameters	641
Trusted Oracle7 Parameters	642
National Language Support Parameters	643
F Contents of the CD-ROM.....	645
SQL Scripts	646
Chapter 9	646
Chapter 10	646
Chapter 16	647
Chapter 25	647

Chapter 27	647
Chapter 28	648
Chapter 29	648
Chapter 32	648
Index	649

Foreword

With the explosion of the Internet phenomena, tens of millions of individuals today get access to and retrieve multimedia data through the World Wide Web. The ease of locating information is provided by sophisticated search mechanisms; the appliance-style interfaces on Web pages is fueling the desire for more data at one's fingertips. And although the personal computer users of today have caught onto online services, the billions of users who can't afford PCs or find them too hard to use will be able to afford and use the next generation of Web appliances. These appliances will offer an even richer multimedia environment than today's PCs for a fraction of the price.

As the Web becomes the next generation "killer app" for the desktop, billions of people who want and need access to data will both increase the amount of data in databases and the number of users on the supporting systems. The result of this increased demand for information is that database server system tuning is increasingly important.

Because relational databases have become the mainstream data storage "vehicle" for modern applications, the flexibility in design and implementation of the underlying database structure, physical file layouts, and tuning parameters can generate wide variations in performance. Although throwing hardware at the problem is often the easy solution, tuning the database application first is usually much more cost effective. Although many books have been written on this subject, you'll find this one to be complete and thorough in its methodologies and execution steps for tuning your Oracle database.

In the time I've known Ed Whalen, he's been extremely thorough and professional with regard to system performance and tuning—and the following pages are pure Ed. I congratulate him on this accomplishment—particularly because he was somehow able to create this volume and work 110 percent on his day job.

Bonnie Crater
Vice President
Oracle Corporation

Acknowledgments

It is not easy acknowledging all the people who have made this book possible. Not only is there the work that went into the book itself, but there is the support and encouragement of friends and family that moved the book forward.

I would like to thank Rosemarie Graham, Todd Bumbalough, Byron Pearce, and especially Alice Martina Smith of Sams for all their help in the development of the book. Without their help, this book could not have been published.

I would also like to thank Ronnie Ward, Gary Stimac, Jim Boak, Keith Carlson, and Mike Perez for making this book possible. Without their support, I would have been unable to write this book.

In writing a book of this type, more than just research is involved. Much of the information I have learned has come from some of the top database performance people in the industry. I would like to recognize the people from whom I have learned much over the years: Brent Schroeder, Karl Haas, Gordon Larimer, Steve DeLuca, Tom Rhodes, Mike Nikolaiev, Jeff Smits, Bernie Luksich, Gina Miscovich, David Simons, Bob Nissen, Bryon Georgson, Marci Frohock, and Jeff Plank.

I would also like to recognize Ken Jacobs, who has represented Oracle for many years on the Transaction Processing Performance Council. I would also like to recognize Hamid Bahadori of Oracle, who has made sure that Oracle always performs well. Other Oracle people for whom I hold the highest respect include Bonnie Crater, Richard French, and Alex Ip.

Writing a book involves a lot of time and effort. I would like to thank my wife, Felicia, for putting up with the sacrifices necessary to write this book.

Finally, I would like to thank David Letterman for providing me with many years of late-night television enjoyment.

The acknowledgments for a book are difficult to write because I am always afraid I have missed someone. If I have, I deeply regret it and apologize.

About the Author

Edward Whalen

Edward Whalen is currently working at Compaq Computer Corporation in the database engineering group. As part of this group, he is responsible for Oracle performance engineering and benchmarking. In this role, he has had much experience in database system design and tuning for optimal performance. During the time he has been in this group, Compaq has published numerous record-breaking TPC-C benchmarks results.

Mr. Whalen is a representative on the Transaction Processing Performance Council, which is responsible for creating and maintaining industry standard database benchmarks. As part of this council, he has participated in the development of several TPC benchmarks and is currently the chairman of the TPC-C maintenance subcommittee.

Mr. Whalen currently resides in Cypress, Texas, with his wife, Felicia; their Border Collies, Pierce (Dash), Chip, Teller, and Ty; their Great Pyrenees, Shasta; and the cats. He is active in many dog-related activities, including dog agility.

He is also a certified EMT and volunteers with the local emergency ambulance service, *Cypress Creek EMS*, where he is a regular on Medic-53, Medic-54, and Medic-55.

Introduction

Database management software and the manipulation of data has evolved to where it touches every aspect of our lives. A day doesn't go by in which we don't access a database. Whether we are withdrawing money from an ATM machine, opening a checking account, or purchasing groceries, every aspect of our lives is affected by databases. Hand in hand with the new power of information comes the frustration of having to wait for data to be retrieved. I'm sure there isn't a person today who hasn't had to wait for a credit card to be approved. Although the speed of computers has been increasing every year, so has the amount of data being manipulated. Amounts of data that several years ago were unheard of are now a daily part of many companies. In years past, databases were used strictly in the realm of big business because large mainframes cost millions of dollars; today, gigabytes of data are being manipulated on the same types of computer you may have in your own home.

No matter how fast new generations of computers get, applications will always be written to take advantage of them. As the cost of storage continues to drop, the amount of data stored will continue to increase. A perfect example of this is the CD-ROM. The advent of the CD-ROM allowed large amounts of data to be inexpensively stored; predictably, many new applications have arisen to take advantage of that technology. These applications are now augmenting written text with video and audio clips. The same type of information revolution is also happening in the database industry.

Oracle already has the capability to store video, documents, and large binary objects in the database and allow quick access to this data. Oracle databases can store hundreds of gigabytes of data and can easily retrieve it; Oracle has the potential of storing *terabytes* of data in a single database in the near future.

What is needed to store this kind of data? You must have a Relational Database Management System (RDBMS) with the following attributes:

- ◆ **Can effectively handle large amounts of data.** The amount of data in entry-level systems is increasing at an amazing rate.
- ◆ **Can manage large numbers of uses.** Contention must be effectively managed and data integrity guaranteed.
- ◆ **Can maintain the required level of security.** With large numbers of users, it is imperative that protected data be secured.
- ◆ **Can consistently perform at the required rates.** To serve a large user community effectively, the system must be able to maintain the performance rate required by the user community.
- ◆ **Is available as needed.** The system may be required to perform in a 7×24 environment (that is, up 24 hours a day, 7 days a week).
- ◆ **Is reliable.** The system is only as good as its reliability. An unreliable system is unacceptable in today's mission-critical environments.

- ◆ **Is serviceable.** The system should be easily maintained and serviced to reduce downtime and boost productivity.
- ◆ **Is within your budget.** A perfect solution that you cannot afford is not a good solution.

This book focuses on engineering the performance of Oracle systems while maintaining the attributes just mentioned. To achieve optimal performance, you must incorporate many factors, including engineering performance into the initial design of the database layout, properly constructing the application code, tuning the database, properly selecting the hardware, designing the network, and choosing the operating system.

What's in this Book?

In this book, I help you build a foundation of understanding. When you finish reading this book, you should have not only an understanding about how to design high-performance applications but also the knowledge about how the system operates and how each component contributes to the performance of the whole. With this philosophy in mind, the book was designed with six parts, each of which has several chapters.

Part I, “Introduction,” lays down a foundation by presenting an overview of the Oracle architecture and products. You will gain an understanding of the design of Oracle and the different features available. I also discuss what it means to have a well-performing system. If you tune a system without having any goals in mind, you usually end in disappointment.

Chapter 4, “Tuning Methodology,” is an overview of my methodology for system performance tuning. Here you will find the way to plan, execute, and analyze changes that have been made and how to determine what the result means.

In Chapter 5, “Benchmarking,” you learn how to set up a test to show whether your system is performing well and where the problem areas are. You also learn about some of the industry standard benchmarks and what they mean to you. You are introduced to the Transaction Processing Performance Council (TCP), their Web site, and how you can use this information.

Chapter 6, “Performance Monitoring Tools,” gives an overview of some of the performance monitoring tools available from Oracle and third-party vendors and describes how they can help you in investigating performance problems.

Chapter 7, “Performance Engineering Starts at the Design Stage,” explains the holistic approach to performance engineering. Performance should be engineered into every aspect of the system, from hardware selection and network configuration to database design and application development.

Part II, “Tuning the Server,” focuses on tuning the RDBMS server. The performance of the system depends on a number of factors. By analyzing each component individually, you can determine where performance can be enhanced.

Chapter 8, “What Affects Oracle Server Performance?” is an overview of what can effect the server performance and where bottlenecks can occur.

Chapter 9, “Oracle Instance Tuning,” reviews the bulk of instance tuning. Items such as memory, I/O, and CPU usage are discussed here. You also get a more in-depth look at some internal Oracle structures.

Chapter 10, “Performance Enhancements,” looks at some of the things you can do in the RDBMS server to improve performance. Some examples are the use of clusters and indexes as well as some of Oracle’s parallel features.

Chapter 11, “Tuning the Server Operating System,” discusses the OS parameters you can tune. Chapter 12, “Operating System-Specific Tuning,” focuses on a few of the 90 platforms Oracle supports.

Chapter 13, “System Processors,” Chapter 14, “Advanced Disk I/O Concepts,” and Chapter 15, “Disk Arrays,” describe some specifics concerning hardware and include detailed explanations about why different hardware features affect RDBMS performance.

Part III, “Configuring the System,” looks at differences in design, tuning, and configuration based on the type of application to be run. This part of the book is designed to give you an understanding of the differences in data access patterns and system loading based on your application profile. In addition to explaining the differences in system workload, the chapters in this part of the book provide solutions to the problems these workloads cause.

Part IV, “Tuning SQL,” and **Part V, “Tuning the Client,”** explore the client. Here, the focus is on the application and the front-end pieces and how to improve the client’s performance. Part IV focuses on the SQL statements used in your application; Part V looks more at the client machines themselves and how you can tune them.

Chapter 24, “What Is a Well-Tuned SQL Statement?” introduces the concept of badly tuned SQL statements and how they can affect the performance of the client and the server. It is here that the importance of a well-tuned SQL statement is discussed. The remainder of Part IV looks at various topics relating to how your SQL statements can be improved and tuned. Included are many tips and hints on features and techniques that can improve your SQL statements.

Part V focuses on the client OS and improvements you can make to streamline the system. The goal of tuning the client OS is to get it out of the way of the client application so that the client application can be more efficient. Chapter 34, “Using GUI Builders,” examines the popular GUI builder products available on the market and how you can tune the resulting application to improve performance.

Part VI, “Tuning the Network,” is what is left over when you have tuned the server and the application. It covers how you know if you have a network problem and how you solve it.

How To Use this Book

To keep the book interesting, I have woven in some personal anecdotes relevant to the subject matter. I hope I have conveyed some of the excitement that comes when you push systems to their limits. Those of us who work in the database performance field constantly push the envelope of technology to achieve new levels of performance previously thought impossible. This kind of experimentation can be very satisfying when everything works well; it can be frustrating when it doesn't.

My hopes are that, having read this book, you have a basic understanding of how the components of the system work together to form the whole. If you have this foundation, I feel confident that you can tackle a performance problem, know what to look for, and know how to fix it. Not all performance problems are alike, nor are all solutions. It is important that you have a basic understanding of what to look for and what the possible solutions are.

To me, performance is one of the most exciting areas of computing. Personal computers now have the computing power reserved for mainframes just a few years ago. Gigabyte hard disks for less than \$300 now make it possible for us to have huge stores of information in our home computers. But performance does not mean buying the biggest and the fastest; you have to get the most out of what you've got. Buying a bigger or a faster server may be a waste if, in fact, the problem is the client machine or the network: a 10-second response time to a query could be caused by a 1-second response from the server and a 9-second wait for the information to be processed in the GUI.

If I have done my job correctly, you should finish this book with the ability to analyze the problem, hypothesize a solution, test that solution, and understand the result. I hope this book gives novices an idea of what performance engineering is all about; seasoned professionals should receive some new insight and ideas.

Introduction

Chapter 1 Introduction to Oracle

- 2** Understanding Terms
- 3** What Is a Well-Tuned System?
- 4** Tuning Methodology
- 5** Benchmarking
- 6** Performance Monitoring Tools
- 7** Performance Engineering Starts at the Design Stage

Part I of this book lays down a foundation by presenting an overview of the Oracle architecture and products. In the first few chapters, you gain an understanding of the design of Oracle and the different features available. You also learn what it means to have a well-performing system. If you tune a system without having any goals in mind, you usually end in disappointment.

Chapter 4, “Tuning Methodology,” is an overview of my methodology for system performance tuning. This chapter presents the way to plan, execute, and analyze changes and determine what the results mean.

In Chapter 5, “Benchmarking,” you learn how to set up a test to show whether your system is performing well and where the problem areas are. You also learn about some of the industry standard benchmarks and what they mean to you. You are introduced to the Transaction Processing Performance Council (TCP), their Web site, and how you can use the information available.

Chapter 6, “Performance Monitoring Tools,” provides an overview of some of the performance monitoring tools available from Oracle and third-party vendors. The chapter explains how these tools can help you investigate performance problems.

Chapter 7, “Performance Engineering Starts at the Design Stage,” explains the holistic approach to performance engineering. Performance should be engineered into every aspect of the system—from hardware selection and network configuration to database design and application development.

Chapter

Introduction to Oracle

To effectively enhance the performance of your Oracle system, it is essential to understand how the product operates. Therefore, I think it appropriate to start this book by reviewing the Oracle architecture. Following the architectural overview, I give an overview of the products and services Oracle provides. This introduction will leave you with an appreciation of the diversity of products Oracle offers.

Oracle Corporation is one of the world's largest vendors of software for managing information. Oracle has over 12,000 employees with offices in 93 countries around the world. One of the reasons Oracle software is so popular is the diversity of platforms it supports. In fact, Oracle software runs on almost every popular computer in the world and is used everywhere from home applications to giant corporations. Because Oracle is expanding into video, audio, and textual data for consumer applications, the company is poised to become the leading software provider for the Information Superhighway.

Oracle's products span many areas beyond the core RDBMS with which we are all familiar. Later in this chapter is an overview of some of the many Oracle products and services available, but first let's look at the core RDBMS.

The Oracle RDBMS (Relational Database Management System) is a product designed to allow simultaneous access into large amounts of stored information. The RDBMS consists of the *database* (the information) and the *instance* (the embodiment of the system). The database consists of both the physical files that reside on the system and the logical pieces such as the database schema. These database files take various forms, as described in the following section. The instance is the method used to access the data and consists of processes and system memory.

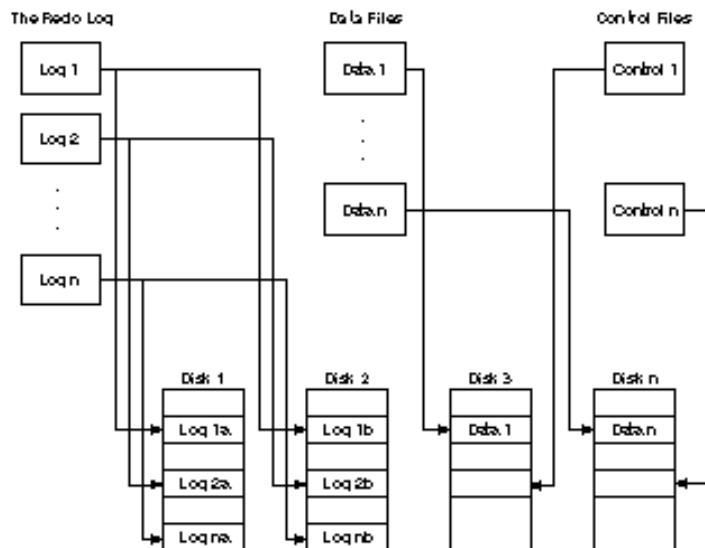
The Database

The Oracle database has both a logical and a physical layer. The physical layer consists of the actual files that reside on the disk; the components of the logical layer map the data to these physical components.

The Physical Layer

The physical layer of the database consists of three types of files: one or more data files, two or more redo log files, and one or more control files. The *data files* store the information contained in the database. The *redo log files* hold information used for recovery in the event of a failure. The *control files* contain information used to start up an instance, such as the location of data files (see Figure 1.1).

Figure 1.1
The Oracle file structure.



Data Files

The data files contain the information stored in the database. You can have as few as one data file or as many as hundreds of data files. The information for a single table can span many data files or many tables can share a set of data files. Spreading tablespaces over many data files can have a significant positive effect on performance, as discussed in later chapters. The number of data files that can be configured is limited by the Oracle parameter `MAXDATAFILES`.

Redo Log Files

The redo log files, known collectively as the *redo log*, store a log of all changes made to the database. This information is used in the event of a system failure to reapply changes that have been made and committed but that may not have been made to the data files. It is essential that the redo log files have good performance and are protected against hardware failures (either through software or hardware fault tolerance). If redo log information is lost, you cannot recover the system.

Control Files

The control files are used to store information such as the locations of data and redo log files; Oracle needs this information to start up the database instance. It is essential that the control files are protected. Oracle provides a mechanism for storing multiple copies of the control files.

The Logical Layer

The logical database consists of the following elements:

- ◆ One or more tablespaces.
- ◆ The database schema, which consists of items such as tables, clusters, indexes, views, stored procedures, database triggers, sequences, and so on.

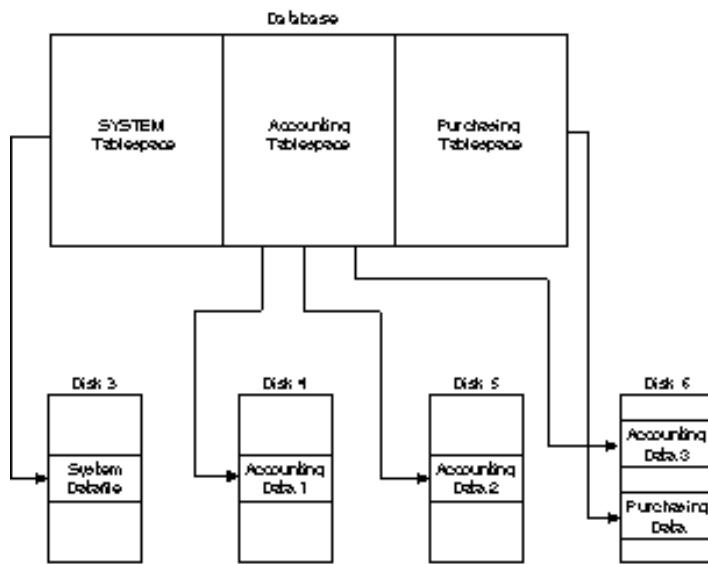
Tablespaces and Data Files

The database is at the highest level and is divided into one or more logical pieces known as *tablespaces*. A tablespace is used to logically group data together. For example, you can have a tablespace for accounting and a separate tablespace for purchasing. Segmenting groups into different tablespaces simplifies the administration of these groups (see Figure 1.2). Tablespaces are made up of one or more data files. By using more than one data file per tablespace, you can spread the data over many different disks to distribute the I/O load and improve performance.

As part of the process of creating the database, Oracle automatically creates the SYSTEM tablespace for you. Although a small database can fit within the SYSTEM tablespace, it is recommended that you create a separate tablespace for user data. The SYSTEM tablespace is where the data dictionary is kept. The data dictionary contains information about tables, indexes, clusters, and so on.

Figure 1.2

The relationship between the database, tablespaces, and data files.



The data files can be operating system files; in the case of some operating systems, the data files can be raw devices. Data files and data access methods are described in detail in Chapter 12, “Operating System-Specific Tuning.”

Database Schema

The *database schema* is a collection of logical structures that define how you see the database’s data. These logical structures are known as *schema objects*. Schema objects are made up of structures such as tables, clusters, indexes, views, stored procedures, database triggers, and sequences.

- ◆ **Table.** A table is the basic logical storage unit in the Oracle database. A table is made up of a table name and rows and columns of data. The columns are defined by name and data type. A table is stored within a tablespace. It is common for many tables to share a tablespace.
- ◆ **Cluster.** A cluster is a set of tables physically stored together as one table. If data in two or more tables is frequently retrieved together and closely related, using a clustered table can be quite efficient. Tables can be accessed separately even though they are part of a clustered table. Because of the structure of the cluster, if related data is accessed simultaneously, it requires much less I/O overhead.
- ◆ **Index.** An index is a structure created to help retrieve data more quickly and efficiently (just as the index in this book allows you to find a particular section more quickly).
- ◆ **Views.** A view is a window into one or more tables. A view does not store any data; it is used for *presentation* of table data. A view can be queried, updated, and deleted as a table without restriction.

Views are typically used to simplify the user's perception of data access by providing information from several tables transparently. Views can also be used to prevent some data from being accessed by the user or to create a join from multiple tables.

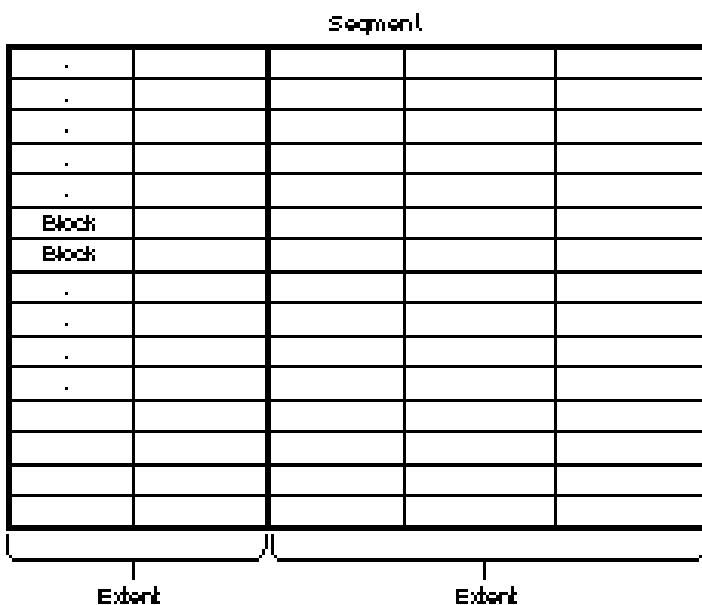
- ◆ **Stored procedures.** Stored procedures are predefined SQL queries stored in the data dictionary designed to allow more efficient queries. By using stored procedures, you can reduce the amount of information that must be passed to the RDBMS and thus reduce network traffic and improve performance.

Segments, Extents, and Data Blocks

Within Oracle, the space used to store data is controlled by the use of logical structures. These structures consist of segments, extents, and data blocks. A *segment* is a set of extents used to store a particular type of data. Segments in turn are made up of collections of pieces called *extents*; in turn, extents are made up of pieces called *data blocks* (see Figure 1.3). A *block* is the smallest unit of storage in an Oracle database. The database block contains header information concerning the block itself as well as the data.

Figure 1.3

Segments, extents, and data blocks.



Note: Within an extent, the data blocks are contiguous.

A **segment** is a group of extents used for storing a particular type of data within the database. An Oracle database can use four different types of segments:

- ◆ *Data segment*: Stores user data within the database.
- ◆ *Index segment*: Stores indexes.
- ◆ *Rollback segment*: Stores rollback information used when data must be rolled back.
- ◆ *Temporary segment*: Temporary segments are created when an SQL statement needs a temporary work area; they are destroyed when the SQL statement is finished. These segments are used during various database operations such as sorts.

Extents are the building blocks used to create segments; extents are made up of data blocks. An extent is used to minimize the amount of wasted (empty) storage. As more and more data is input into tablespaces in your database, the extents used for storing that data can grow or shrink depending on need. In this manner, many tablespaces can share the same storage space without preallocating the divisions between those tablespaces.

At tablespace creation time, you can specify the minimum number of extents to allocate as well as the number of extents to add at a time when that allocation has been used up. This arrangement gives you efficient control over the space used in your database.

Data blocks are the smallest pieces that make up an Oracle database; they are physically stored on disk. Although the data block in most systems is 2K (2,048 bytes), you can change this size for efficiency depending on your application or operating system. Sizing the data block is described in detail in Chapter 10, “Performance Enhancements.”

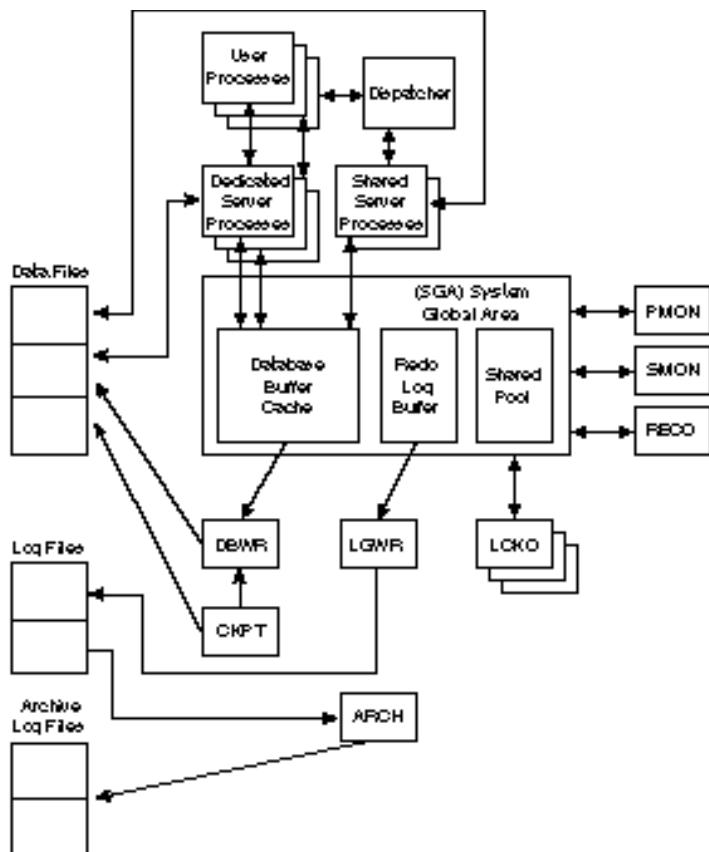
The Oracle Instance

The Oracle instance consists of the Oracle processes and shared memory necessary to access information in the database. The instance is made up of the user processes, the Oracle background processes, and the shared memory used by these processes (see Figure 1.4).

The following sections look at the various pieces that make up the Oracle instance, starting with the shared memory and continuing with the various Oracle processes.

The Oracle Memory Structure

Oracle uses shared memory for several purposes, including caching of data and indexes as well as storing shared program code. This shared memory is used for several functions and is broken into various pieces, or *memory structures*. The basic memory structures associated with Oracle are the System Global Area (SGA) and the Program Global Area (PGA).

Figure 1.4*The Oracle instance.*

System Global Area (SGA)

The SGA is a shared memory region Oracle uses to store data and control information for one Oracle instance. The SGA is allocated when the Oracle instance starts; it is deallocated when the Oracle instance shuts down. Each Oracle instance that starts has its own SGA. The information in the SGA is made up of the database buffers, the redo log buffer, and the shared pool; each has a fixed size and is created at instance startup.

- ◆ **Database buffer cache.** The *buffer cache* stores the most recently used data blocks. These blocks can contain modified data that has not yet been written to disk (sometimes known as *dirty blocks*), blocks that have not been modified, or blocks that have been written to disk since modification (sometimes known as *clean blocks*). Because the buffer cache keeps blocks based on a most recently used algorithm, the most active buffers stay in memory to reduce I/O and improve performance.

- ◆ **Redo log buffer.** The redo log buffer of the SGA stores redo entries or a log of changes made to the database. The redo log buffers are written to the redo log as quickly and efficiently as possible. Remember that the redo log is used for instance recovery in the event of a system failure.
- ◆ **Shared pool.** The shared pool is the area of the SGA that stores shared memory structures such as shared SQL areas. Shared SQL areas store the parse tree and the execution plan for every unique SQL statement. If multiple applications issue the same SQL statement, the shared SQL area can be accessed by each of them to reduce the amount of memory needed and to reduce the processing time used for parsing and execution planning.

Program Global Area (PGA)

The PGA is a memory area that contains data and control information for the Oracle server processes. The amount and content of the PGA depends on the Oracle server options you have installed. The PGA is made up of the following components:

- ◆ **Stack space.** This is the memory that holds the session variables, arrays, and so on.
- ◆ **Session information.** If you are not running the multithreaded server, the session information is stored in the PGA. If you are running the multithreaded server, the session information is stored in the SGA.
- ◆ **Private SQL area.** This is an area in the PGA where information such as binding variables and runtime buffers are kept.

Processes

The term *process* is used in this book to describe a *thread of execution*, or a mechanism that can execute a set of code. In many operating systems, traditional “processes” have been replaced with “threads” or “lightweight processes.” In this book, the term *process* refers to the mechanism of execution and can refer to either a traditional process or a thread.

The Oracle RDBMS uses two types of processes: the user processes and the Oracle processes (also known as *background processes*).

User Processes

User, or client, processes are the user’s connections into the RDBMS system. The user process manipulates the user’s input and communicates with the Oracle server process through the Oracle program interface. The user process is also used to display the information requested by the user and, if necessary, can process this information into a more useful form.

Oracle Processes

The Oracle processes perform functions for the users. The Oracle processes can be split into two groups: server processes (which perform functions for the invoking process) and background processes (which perform functions on behalf of the entire RDBMS).

Server Processes (Shadow Processes)

The server processes, also known as *shadow processes*, communicate with the user and interact with Oracle to carry out the user's requests. For example, if the user process requests a piece of data not already in the SGA, the shadow process is responsible for reading the data blocks from the data files into the SGA. There can be a one-to-one correlation between the user processes and the shadow processes (as in a dedicated server configuration); although one shadow process can connect to multiple user processes (as in a multithreaded server configuration), doing so reduces the utilization of system resources.

Background Processes

Background processes are the Oracle processes used to perform various tasks within the RDBMS system. These tasks vary from communicating with other Oracle instances, performing system maintenance and cleanup, to writing dirty blocks to disk. Here are brief descriptions of the nine Oracle background processes:

- ◆ **DBWR (Database Writer).** DBWR is responsible for writing dirty data blocks from the database block buffers to disk. When a transaction changes data in a data block, it is not necessary for that data block to be immediately written to disk. Because of this, the DBWR can write this data out to disk in a manner that is more efficient than writing when each transaction completes. Usually, the DBWR writes only when the database block buffers are needed for data to be read in. When data is written out, it is done in a least recently used fashion. For systems in which Asynchronous I/O (AIO) is available, there should be only one DBWR process. For systems in which AIO is not available, performance can be greatly enhanced by adding more DBWR processes.
- ◆ **LGWR (Log Writer).** The LGWR process is responsible for writing data from the log buffer to the redo log.
- ◆ **CKPT (Checkpoint process).** The CKPT process is responsible for signaling the DBWR process to perform a checkpoint and to update all the data and control files for the database to indicate the most recent checkpoint. A *checkpoint* is an event in which all modified database buffers are written to the data files by the DBWR. The CKPT process is optional. If the CKPT process is not present, the LGWR process assumes these responsibilities.
- ◆ **PMON (Process Monitor).** PMON is responsible for keeping track of database processes and cleaning up if a process prematurely dies (PMON cleans up the cache and frees resources that may still be allocated). PMON is also responsible for restarting any dispatcher processes that may have failed.

- ◆ **SMON (System Monitor).** SMON performs instance recovery at instance startup. This includes cleaning up temporary segments and recovering transactions that have died because of a system crash. SMON also defragments the database by coalescing free extents within the database.
- ◆ **RECO (Recovery process).** RECO is used to clean up transactions that were pending in a distributed database. RECO is responsible for committing or rolling back the local portion of the disputed transactions.
- ◆ **ARCH (Archiver process).** ARCH is responsible for copying the online redo log files to archival storage when they become full. ARCH is active only when the RDBMS is operated in ARCHIVELOG mode. When a system is not operated in ARCHIVELOG mode, it may not be possible to recover after a system failure. You should always operate in ARCHIVELOG mode.
- ◆ **LCKn (Parallel Server Lock processes).** Up to 10 LCK processes are used for interinstance locking when the Oracle Parallel Server option is used.
- ◆ **Dnnn (Dispatcher processes).** When the Multithreaded Server option is used, at least one dispatcher process is used for every communications protocol in use. The dispatcher process is responsible for routing requests from the user processes to available shared server processes and back.

How Transactions Work

To give you a better idea of how Oracle operates, this section analyzes a sample transaction. Throughout this book, the term *transaction* is used to describe a logical group of work. This group of work can consist of one or many SQL statements and must end with a commit or a rollback. Because this example assumes a client/server application, SQL*Net is necessary. The following steps are executed to complete the transaction:

1. The application processes the user input and creates a connection to the server through SQL*Net.
2. The server picks up the connection request and creates a server process on behalf of the user.
3. The user executes an SQL statement and commits the transaction. In this example, the user changes the value of a row in a table.
4. The server process takes this SQL statement and checks the shared pool to see whether there is a shared SQL area that has this identical SQL statement. If it finds an identical shared SQL area, the server process checks to see whether the user has access privileges to the data. If the user has access privileges, the server process uses the shared SQL area to process the request. If a shared SQL area is not found, a new shared SQL area is allocated, the statement is parsed, and then it is executed.
5. The server process retrieves the data from the SGA (if present) or retrieves it from the data file into the SGA.

6. The server process modifies the data in the SGA. Remember that the server processes can only read from the data files. Because the transaction has been committed, the LGWR process immediately records the transaction in the redo log file. At some later time, the DBWR process writes the modified blocks to permanent storage.
7. If the transaction is successful, a completion code is returned across the network to the client process. If a failure has occurred, an error message is returned.

NOTE: A transaction is not considered committed until the write to the redo log file has been completed. This arrangement ensures that, in the event of a system failure, a committed transaction can be recovered. If a transaction has been committed, it is guaranteed to be “cast in stone.”

While transactions are occurring, the Oracle background processes are all doing their jobs, keeping the system running smoothly. Keep in mind that while this process is going on, hundreds of other users may be doing similar tasks. It is Oracle’s job to keep the system in a consistent state, to manage contention and locking, and to perform at the necessary rate.

This overview is intended to give you an understanding of the complexity and amount of interaction involved in the Oracle RDBMS. As you look at the details of tuning the server processes and applications later in this book, you can use this overview as a reference to the basics of how the Oracle RDBMS operates. Because of the differences in operating systems, minor variances in different environments will be discussed individually.

Your introduction to Oracle continues with a look at the different products Oracle offers.

Oracle Products

Oracle produces a wide range of products and services, many of which you may not be aware of. Of course, the product you *are* familiar with is the RDBMS product that supports over 90 platforms. Within the RDBMS server product itself are many options to choose from: the Procedural option, parallel features such as the Parallel Query option, and (in some platforms) features such as Oracle Parallel Server. Oracle has an impressive set of development tools, such as Developer/2000, designed to create applications for the Windows, Macintosh, and Motif environments. Oracle also has traditional character-based applications and applications such as the Oracle Financials suite. All these products are backed by a rich set of support options. The following sections give a general overview of the Oracle products, starting with the RDBMS product itself.

Oracle RDBMS Products

The Oracle RDBMS is the core of Oracle’s product set. Many—if not all—the products Oracle offers use the RDBMS in one way or another. Following is a list of some of the additional features available within the core RDBMS.

PL/SQL

PL/SQL is commonly known as the Procedural option (Oracle's procedural language extension to SQL). PL/SQL adds the procedural functionality of a structured programming language to SQL. PL/SQL allows code to be stored in the database to reduce network traffic between applications and the database. This reduction in network traffic can help improve performance. Another feature of PL/SQL is its ability to control data access by allowing the users access to only PL/SQL statements; this arrangement distances the user from the data layer. To reduce network contention, users can send blocks of PL/SQL statements to the database instead of sending individual SQL statements.

SQL*Net

SQL*Net is the interface used to connect user and server processes on different machines on a network or between user processes on the same machine through a shared dispatcher process. SQL*Net includes the physical network connection through a variety of different protocols. For example, with SQL*Net and TCP/IP on your PC, you can communicate to Oracle on any platform running SQL*Net TCP/IP. By adhering to standards, any of the supported Oracle platforms can communicate with any other platform, provided that they are speaking SQL*Net with the same protocol.

The application programmer needs only to program to the SQL*Net API to provide this transparent connection. If the network protocol were to change, you only have to relink the application with the proper SQL*Net protocol(s).

Distributed Option

The Distributed option allows you to divide data among many different components in a network; using a two-phase commit, you can transparently access the data in a consistent manner. The Distributed option also allows transparent access to remote data. A *two-phase commit* is a mechanism in which data from separate machines can be manipulated together and is guaranteed to either be committed or rolled back together. In conjunction with the Procedural option, the Distributed option allows for read-only copies (known as *snapshots*) of tables to be replicated. Snapshots can be queried but not updated; they are periodically updated by the master table.

Parallel Server Option

The Oracle Parallel Server option allows for two or more systems to access the same database tables by using a shared-disk mechanism. The option can be used both for increasing performance by adding more computers and for fault tolerance. If a hardware failure occurs in a parallel server configuration, the other members of the cluster continue running and automatically roll back any transactions that were pending on the failed system. The Parallel Server option is not available on all platforms; it requires special hardware and operating system support.

Oracle Symmetric Replication

Symmetric Replication is a new option introduced in Oracle version 7.2. It allows for replication of tables in a configuration so that all copies of the table can be modified. Data consistency is guaranteed through a complex set of rules. Full tables or subsets of tables can be replicated.

Trusted Oracle

Trusted Oracle is Oracle's secure database management system product, designed to provide the high level of security required by companies and government agencies that handle sensitive data. Trusted Oracle is compatible with all the base Oracle products and supports all the functionality of standard Oracle. Trusted Oracle enforces mandatory access control across a wide range of secure operating systems. Trusted Oracle is not available on all platforms; it reduces performance because of the extra overhead associated with access control.

Parallel Query Option

The Parallel Query option is a set of features that allows a single query or operation to be divided among many processes. This feature can significantly improve performance in both multiprocessor and single-processor environments. Performance is increased by ensuring that the system CPUs stay busy doing work while other processes are waiting for I/Os to complete. The Parallel Query option is made up of the following components:

- ◆ **Parallel Query:** Allows SQL queries to be split into different components and to execute in parallel.
- ◆ **Parallel Recovery:** Performs instance recovery in parallel, greatly reducing the time it takes to recover from a system failure.
- ◆ **Parallel Index Creation:** Allows for indexes to be built in parallel, reducing index-creation time.
- ◆ **Parallel Load:** If you can separate load data into multiple files, you greatly reduce load time by allowing parallel load threads.
- ◆ **Parallel Backup:** Allows multiple backup streams from the same table.

Server Manager

The Oracle Server Manager provides a GUI tool for the administration of your Oracle RDBMS. Server Manager is available for Windows and Motif; a non-GUI, character-based version of this application is also available. Server Manager is intended to eventually replace SQL*DBA.

Oracle Workgroup Server

Workgroup Server is a powerful combination of the Oracle version 7 RDBMS and a complete set of point-and-click GUI tools that simplify database management. The Oracle version 7

Workgroup Server is part of the Workgroup/2000 suite of client/server tools. Workgroup Server comes preconfigured and is easy to install. The core RDBMS shipped with Workgroup Server is identical to the RDBMS shipped with the standard product. Some high-end add-ons (such as the Parallel Query option and the Distributed option) are not available on the Oracle Workgroup Server.

Workgroup Server is available on all major PC servers, including NetWare, NetWare NT, Solaris x86, SCO UNIX, Novell NetWare, and OS/2. As the PC server platform continues to increase in performance and reliability, the Oracle Workgroup Server provides an unbeatable value.

Here are some of the standard tools included with Workgroup Server:

- ◆ **Database Manager:** Manages the instance, including startup, shutdown, and configuration.
- ◆ **User Manager:** Manages user accounts and access security.
- ◆ **Database Expander:** Expands the database as needed without shutting the system down.
- ◆ **Object Manager:** Displays information about the database tables, indexes, and synonyms. Tables, indexes, and synonyms can be created from within the Object Manager.
- ◆ **Miscellaneous tools:** Tools such as Import, Export, Loader, and so on are also included.

The tools provided with Workgroup Server have the same look and feel whether your platform is a Windows NT system or UNIX system using X-Windows.

Personal Oracle for Windows

The newly released product, Personal Oracle for Windows, provides a version of the Oracle RDBMS for Microsoft Windows. With Personal Oracle for Windows, you can take Oracle with you on your notebook computer and use SQL*Net to connect to any other Oracle system on your network. Personal Oracle for Windows includes some of the same management tools shipped with Oracle Workgroup Server.

Oracle Development Tools

Oracle offers a rich set of development tools for use in GUI environments as well as character-based applications. These tools offer several approaches to application development based on the end user's needs. Here's a list of the development tools available:

- ◆ **Developer/2000 and Designer/2000:** Developer/2000 and Designer/2000 are the evolution of the Oracle CDE (Cooperative Development Environment) tools.

Developer/2000 is a set of tools that allows the developer to create an application and roll it out in Windows, Macintosh, Motif, and character mode. Developer/2000 incorporates graphics and images as well as support for multimedia objects such as video and sound in a variety of standard formats. Designer/2000 is a set of design tools that assists you with process and data modeling; you can use Designer/2000 to provide input into the Developer/2000 system. Designer/2000 can be used to develop the fundamental models that are the foundation for your business processes.

- ◆ **Power Objects:** Power Objects is a lightweight GUI development tool designed to allow quick development of applications; Power Objects uses relatively few system resources. Power Objects is similar to Developer/2000 in concept—but without a lot of its features. Power Objects is available for Windows, Macintosh, and OS/2.
- ◆ **Objects for OLE:** Objects for OLE is a set of tools that allows you to link OLE-compliant applications to an Oracle RDBMS. This tool provides a quick and easy way to take advantage of the power of applications such as spreadsheets. Objects for OLE also allows for easy linking of database tables into word processing documents.

Oracle Applications

Oracle has many applications built around the Oracle RDBMS system. The most well-known of these are the Oracle Financial applications. Oracle also offers many more applications including Oracle Office, Oracle Media Server, and Oracle ConText.

Oracle Financial Applications

Oracle provides a rich set of financial applications used extensively in businesses around the world. These applications are sometimes known collectively as *Oracle Financials*. The Oracle Financials application set is built around the Oracle RDBMS and includes applications such as Oracle Human Resources, Oracle Payroll, Oracle Payables, Oracle Receivables, Oracle General Ledger, Oracle Assets, Oracle Inventory, Oracle Order Entry, Oracle Purchasing, Oracle Bills of Material, Oracle Engineering, Oracle Master Scheduling, Oracle Work in Progress, Oracle MRP, Oracle Capacity, Oracle Project Costing, and Oracle Project Billing.

These applications are used to run many of the world's mission-critical businesses. These applications are available on most platforms.

Oracle Office

Oracle Office provides a solution for today's office needs. Office provides a GUI office environment as well as several mail gateway products. The Oracle Office environment includes the following components:

- ◆ Enterprise-wide messaging
- ◆ Enterprise-wide scheduling

- ◆ Personal calendar
- ◆ Personal organizer
- ◆ Proofreader

These components, in conjunction with other Oracle products, can be used to seamlessly integrate office and database applications and functionality.

Media Server

Oracle Media Server provides high-performance, scaleable, and reliable multimedia library functions on a wide variety of general-purpose systems. Media Server handles the storage, retrieval, and management of movies, music, photographs, and text articles.

ConText

When integrated with any text system, Oracle ConText can analyze and provide text filtering and text reduction for speed reading and summary viewing. Oracle ConText returns detailed grammatical assessments of the text it processes, checking for grammatical errors and rating the quality and style of the writing.

Oracle WebServer

The newly introduced Oracle WebServer is designed to provide front-end services that allow World Wide Web access to an Oracle database. This product allows Web users to retrieve information directly from an Oracle database rather than from traditional flat files. This product can be used to enhance the performance and functionality of your Web server. Performance is enhanced through the use of indexes and data caching. With the flexibility of the Oracle RDBMS, the functionality of your Web server can be enhanced through the use of language-sensitive context and other features.

Oracle Services

Oracle Corporation complements all its products with a wide variety of services. These services include worldwide consulting, support, and education.

Oracle Consulting

Oracle has a staff of more than 3,000 professionals highly skilled in Oracle applications and products. These specialists can work at your site to ensure quick deployment and support of Oracle applications. Oracle consulting supports the entire life cycle of your application: from re-engineering to deployment and administration.

Oracle Support

Oracle's worldwide support organization provides comprehensive product support as well as multivendor support. Oracle has cooperative support agreements with key vendors to provide mutual support of customer environments. This arrangement enables quick responses and solutions for multivendor issues.

Oracle Education

To round out Oracle's services, the company offers a wide range of educational programs from computer-based training to advanced classes taught at many Oracle training centers or at your site. The hands-on courses are designed to have you working with Oracle products quickly. Oracle also offers custom classes designed around your needs. Oracle Education offers many options, including these:

- ◆ **Classroom training.** These classes are offered around the world at Oracle training facilities.
- ◆ **Onsite training.** Oracle Education can come to your site for training.
- ◆ **Media-based training.** This includes options such as computer-based training (CBT), CD-ROM, and video training.
- ◆ **The Oracle channel.** This new option allows for face-to-face contact with instructors and other students using satellite connections.

Summary

This chapter provided you with a sample of the wide range of products and services Oracle offers. I hope it didn't sound too much like an advertisement for Oracle, but I believe that you should have an idea of the broad range of products and services available. Many of these products are not mentioned anywhere else in this book; after all, you bought this book to learn how to improve performance.

You should realize that Oracle is constantly moving into new areas as well as improving its existing products. As computers get faster and less expensive in the next few years, I'm sure you will see the RDBMS evolve to accommodate these changes.

Chapter **2**

Understanding Terms

Chapter 1 gave an introduction to Oracle. That introduction gave you a better understanding of how the RDBMS engine itself works as well as some insight into how you can use that knowledge to effectively design an optimal system.

In following the philosophy of building on a foundation of knowledge, this chapter discusses some important terms used throughout the book. Because these terms and descriptions are used frequently, I think it is appropriate to provide these definitions early in the book.

This chapter is split into several sections. The first section describes some terms frequently used when discussing database engineering and databases. The second section involves further explanations of some of the internal workings of an RDBMS. The final section describes some of the business models RDBMS systems are commonly used for. The last section reviews the unit conversions used in the book.

Terms

Before starting any explanation of the functions of Oracle, I would like to list a few terms that are used throughout the book. These terms are frequently used in the industry; for newcomers, clear definitions are not always given. Because many of these terms are not related, they are listed in alphabetical order. These and many other terms are also listed in the glossary in Appendix D.

ad-hoc. From the Latin *this is*. This term is used to describe an impromptu or spontaneous action. Most commonly used in terms of an *ad-hoc query* to mean an *impromptu, simple query*.

Asynchronous I/O (AIO). Asynchronous I/O allows a process to submit an I/O and not have to wait for the response. Later, when the I/O is completed, an interrupt occurs or the process can check to see whether the I/O has completed. By using Asynchronous I/Os, the DBWR can manage multiple writes at once so that it is not starved waiting for I/Os to complete.

block. The smallest unit of storage in an Oracle database. The database block contains header information concerning the block itself as well as the data.

buffer. An amount of memory used to store data. A buffer stores data that is about to be used or that has just been used. In many cases, buffers are in-memory copies of data that is also on disk. Buffers can be used as a copy of data for quick read-access, they can be modified and written to disk, or they can be created in memory as temporary storage.

In Oracle, the database buffers of the SGA store the most recently used blocks of database data. The set of database block buffers is known as the *database buffer cache*. The buffers used to temporarily store redo entries until they can be written to disk are known as the *redo log buffers*.

A *clean buffer* is a buffer that has not been modified. Because this buffer has not been changed, it is not necessary for the DBWR to write this buffer to disk. A *dirty buffer* is a buffer that has been modified. It is the job of the DBWR to eventually write all dirty block buffers out to disk.

cache. A storage area used to provide fast access to data. In hardware terms, the cache is a small (relative to main RAM) amount of memory that is much faster than main memory. This memory is used to reduce the time it takes to reload frequently used data or instructions into the CPU. CPU chips themselves contain small amounts of memory built in as a cache.

In Oracle, the block buffers and shared pool are considered caches because they are used to store data and instructions for quick access. Caching is very effective in reducing the time it takes to retrieve frequently used data.

Caching usually works using a *least recently used* algorithm. Data that has not been used for a while is eventually released from the cache to make room for new data. If data is requested and is in the cache (a phenomenon called a *cache hit*), the data is retrieved from the cache, avoiding having to retrieve it from memory or disk. Once the data has been accessed again, it is marked as recently used and put on the top of the cache list.

checksum. A number calculated from the contents of a storage unit such as a file or data block. Using a mathematical formula, the checksum number is generated from data. Because it is highly unlikely that data corruption can occur in such a way that the checksum would remain the same, checksums are used to verify data integrity. Beginning with Oracle version 7.2, checksums can be enabled on both data blocks and redo blocks.

concurrency. The ability to perform many functions at the same time. Oracle provides for concurrency by allowing many users to access the database simultaneously.

contention. A term usually used to describe a condition that occurs when two or more processes or threads attempt to obtain the same resource. The results of contention can vary depending on the resource in question.

cursor. A handle to a specific private SQL area. You can think of a cursor as a pointer to or a name of a particular private SQL area.

data dictionary. A set of tables Oracle uses to maintain information about the database. The data dictionary contains information about tables, indexes, clusters, and so on.

DDL (Data Definition Language) commands. The commands used in the creation and modification of schema objects. These commands include the ability to create, alter, and drop objects; grant and revoke privileges and roles; establish auditing options; and add comments to the data dictionary. These commands are all related to the management and administration of the Oracle database. Before and after each DDL statement, Oracle implicitly commits the current transaction.

DML (Data Manipulation Language) commands. The commands that allow you to query and modify data within existing schema objects. Unlike the DDL commands, a commit is not implicit. DML statements consist of `DELETE`, `INSERT`, `SELECT`, and `UPDATE` statements; `EXPLAIN PLAN` statements; and `LOCK TABLE` statements.

DBA (Database Administrator). The person responsible for the operation and configuration of the database. The DBA is the person responsible for the performance of the database. The DBA is charged with keeping the database operating smoothly, ensuring that backups are done on a regular basis (and that the backups work), and installing new software. Other responsibilities may include planning for future expansion and disk space needs; creating databases and tablespaces; adding users and maintaining security; and monitoring the database and retuning it as necessary. Large installations may have teams of DBAs to keep the system running smoothly; alternatively, the tasks may be segmented among the DBAs.

disk array. A set of two or more disks that can appear to the system as one large disk. A disk array can be either a software or a hardware device.

dynamic performance tables. Tables created at instance startup and used to store information about the performance of the instance. This information includes connection information, I/Os, initialization parameter values, and so on.

function. A set of SQL or PL/SQL statements used together to execute a particular function. Procedures and functions are identical except that functions always return a value (procedures do not). By processing the SQL code on the database server, you can reduce the amount of instructions sent across the network and returned from the SQL statements.

logical disk. A term used to describe a disk that is in reality two or more disks in a hardware or software disk array. To the user, it appears as one large disk when in reality it is two or more striped physical disks.

package. A collection of related, stored procedures or functions grouped together.

paging. An operating system function used to copy virtual memory between physical memory and the paging file (*see* virtual memory, later in this list). Paging is used when the amount of virtual memory in use has exceeded the amount of physical memory available. Paging is an expensive task in terms of performance and should be avoided if possible.

physical memory. The actual hardware RAM (Random Access Memory) available in the computer for use by the operating system and applications.

procedure. A set of SQL or PL/SQL statements used together to execute a particular function. Procedures and functions are identical except that functions always return a value (procedures do not). By processing the SQL code on the database server, you can reduce the amount of instructions sent across the network and returned from the SQL statements.

program unit. In Oracle, the term used to describe a package, a stored procedure, or a sequence.

random I/O. Occurs when data is accessed on a disk drive in no specific order. Random I/O typically creates significant disk head movement.

read consistency. An attribute used to ensure that, during a SQL statement, data returned from Oracle is consistent. Oracle uses the rollback segments to ensure read consistency.

scalability. Typically used in association with multiprocessor or cluster configurations. The scalability of the additional component refers to the performance gain obtained by adding that component. A perfectly scalable solution gives double the performance when you add a second component.

For example, if you have an SMP machine with a measured performance of 1.0 (normalized), add a second CPU, and get a performance of 1.9, the scalability of adding the second CPU is 1.9 or 90 percent. This term is used quite frequently in hardware and software manufacturer's literature when marketing multiprocessor or clustered solutions.

schema. A collection of objects associated with the database.

schema objects. Abstractions or logical structures that refer to database objects or structures. Schema objects consist of such things as clusters, indexes, packages, sequences, stored procedures, synonyms, tables, views, and so on.

sequential I/O. Occurs when data is accessed on a disk drive in order. Sequential I/O typically causes very little disk head movement.

SGA. See System Global Area.

swapping. An operating system function similar to paging; used to copy virtual memory between physical memory and the paging file (*see* virtual memory, later in this list). Swapping is almost identical to paging except that swapping is done on a process basis and paging is done on a memory-page basis. Swapping is used when the amount of virtual memory in use has exceeded the amount of physical memory available. Swapping is quite expensive in terms of performance and should be avoided if possible.

System Global Area. The SGA is a shared memory region Oracle uses to store data and control information for a single Oracle instance. The SGA is allocated when the Oracle instance starts; it is deallocated when the Oracle instance shuts down. Each Oracle instance that starts has its own SGA. The information in the SGA is made up of the database buffers, the redo log buffer, and the shared pool; each has a fixed size and is created at instance startup.

trigger. A mechanism that allows you to write procedures that are automatically executed whenever an `INSERT`, `UPDATE`, or `DELETE` statement is executed on a table or view. Triggers can be used to enforce integrity constraints or automate some other custom function.

virtual memory. The memory that can be used for programs in the operating system. To overcome the limitations associated with insufficient physical memory, virtual memory allows programs to run that are larger than the amount of physical memory in the system. When there is not enough physical memory in the system, these programs are copied from RAM to a disk file called a *paging* or *swap file*. This arrangement allows small systems to run many programs. There is a performance penalty you pay when the computer pages or swaps.

transaction. A logical unit of work consisting of one or more SQL statements, ending in a commit or a rollback. Performance measurements, as described in Chapter 5, "Benchmarking," often use the number of transactions per second as the performance metric.

RDBMS Functionality

Chapter 1 reviewed the Oracle architecture. It looked at some of the functions the architecture provides for. In this chapter, I want to take some of these concepts a bit further and explain why they are needed and what they do.

Each of these concepts is crucial to the function of the whole RDBMS system. Each has its own unique characteristics and functionality that you can take advantage of in designing an optimal system.

If the RDBMS is to operate, you must provide for certain functions. Among these are data integrity, recovery from failure, error handling, and so on. The following sections list and describe some of these functions.

Checkpoint

Oracle uses either the CKPT background process or the LGWR process to signal a checkpoint. But what is a checkpoint and why is it necessary?

Because all modifications done to data blocks are done on the block buffers, there are changes to data in memory that are not necessarily reflected in the blocks on disk. Because caching is done using a least recently used algorithm, if a buffer is constantly modified, it is always marked as recently used and is therefore unlikely to be written out by the DBWR.

A checkpoint is used to ensure that these buffers are written to disk by forcing all dirty buffers to be written out on a regular basis. This does not mean that all work stops during a checkpoint; the checkpoint process has two methods of operation: the fast checkpoint and the normal checkpoint.

In the normal checkpoint, the DBWR merely writes out a few more buffers every time it is active. This type of checkpoint takes much longer but has less of an effect on the system. In the fast checkpoint, the DBWR writes a large number of buffers at the request of the checkpoint each time it is active. This type of checkpoint completes much quicker and is more efficient in terms of I/Os generated, but it has a greater effect on system performance at the time of the checkpoint.

You can use the time between checkpoints to improve instance recovery. Frequent checkpoints reduce the time required to recover in the event of a system failure. A checkpoint automatically occurs at a log switch.

Logging and Archiving

The redo log records all changes made to the Oracle database. The purpose of the redo log is to ensure that, in the event of the loss of a data file caused by some sort of system failure, the

database can be recovered. By restoring the data files back to a known good state from backups, the redo log files (including the archive log files) can replay all the transactions to the restored data file, thus recovering the database to the point of failure.

When a redo log file is filled in normal operation, a log switch occurs and the LGWR process starts writing to a different redo log file. When this switch occurs, the ARCH process copies the filled redo log file to an *archive log file*. When this archive is complete, the redo log file is marked as available. It is critical that this archive log file be safely stored because it may be needed for recovery.

Later chapters revisit the concept of the redo log files and the archive log processes in terms of optimizing the log performance.

NOTE: Remember that a transaction has not been committed until the redo log file has been written. Slow I/Os to the redo log files can slow down the entire system.

Business Models

In Part III of this book, “Configuring the System,” you configure for various business models. That part of the book looks at OLTP, batch processing, decision support, data warehousing, and BLOBs. OLTP and batch processing are perhaps the most common ways RDBMS products are used today, but decision support systems are growing rapidly. The following sections describe each of these systems.

OnLine Transaction Processing (OLTP)

OLTP is perhaps the most familiar type of information processing. OLTP stands for OnLine Transaction Processing and is just that: OLTP describes the business model of order entry, sales, data retrieval, and so on in an online fashion. In other words, OLTP describes anyone who is processing transactions in an interactive manner.

Typical attributes include a high number of users accessing data in a concurrent manner with a strict response time criteria and no tolerance for downtime.

Batch Processing

Batch processing is information processing done offline. These applications usually involve heavy load activity and long processing times. An example is the processing of all orders taken during the day.

The attributes of batch processing include strict criteria for load time and long processing times.

Decision Support

Decision support activities involve extremely large queries over large amounts of data for the purpose of making an informed business decision. The reduction in cost of disk storage and improvements in CPU speeds has caused decision support systems to grow in popularity.

Attributes of decision support systems include large queries over large amounts of data. Decision support applications are now taking advantage of GUIs to enhance the interpretation of the output data.

Data Warehousing

The data warehouse system is a large database that holds information integrated from the organization's operational databases. The concept of the data warehouse is new and gaining in popularity as a way of pulling together all the information in the organization.

In some respects, you can think of data warehousing as the integration of OLTP and decision support systems for the entire organization.

Attributes of the data warehousing system include enormous amounts of data with both online and decision support functions.

Binary Large Objects (BLOBs)

BLOBs have become more popular with the emergence of client/server systems. A BLOB is simply a large binary object stored in the database. A BLOB can be an image, audio file, or a full-length movie. As you see in Part III of this book, "Configuring the System," there are ways to improve performance in the system that is storing BLOBs.

Attributes of BLOB systems include large amounts of data and (in some cases) the requirement of a continuous data stream.

Unit Conversions

Most people are familiar with the units of measurement commonly used in the computer industry. But to ensure that all readers of this book start out on the same foot, the following sections give quick overviews of these terms.

Powers of 10

Here is a quick review for those who may have forgotten these conversions:

<i>Power of 10</i>	<i>Unit Name</i>	<i>Actual Value</i>
10^{12}	Tera	1,000,000,000,000
10^9	Giga	1,000,000,000
10^6	Mega	1,000,000
10^3	Kilo	1,000
10^0		
10^{-3}	milli	0.001
10^{-6}	micro	0.000001
10^{-9}	nano	0.000000001
10^{-12}	pico	0.000000000001

These are the typical ranges of numbers used in computer technology, although you don't really see *pico* referred to very often. Speeds in terms of picoseconds are useful only if you are looking at the internal operations of the CPU chips themselves. At the speed of light, it takes approximately 0.3 picosecond to travel 1 millimeter.

Storage Units

Data is stored in the computer in a binary form. The units used to refer to this binary data are as follows:

<i>Term</i>	<i>Definition</i>	<i>Comment</i>
bit	The smallest unit of data storage	A bit is either a 1 or a 0.
nibble	4 bits	This term is not commonly used.
byte	8 bits	The most commonly used storage unit.
word	An architecture-dependent term	On some systems, a word is 16 bits; on others, a word is 32 or 64 bits.

continues

<i>Term</i>	<i>Definition</i>	<i>Comment</i>
kilobyte, Kbyte, KB		Even though the term <i>Kilo</i> usually means 1,000, in computer terms (because we like powers of 2), a kilobyte is actually 1,024 bytes.
Megabyte, Mbyte, MB		Just as with the kilobyte, the term megabyte refers to 1,024KB or 1,048,576 bytes.
gigabyte, Gbyte, GB		A gigabyte is 1,024 megabytes or 1,073,741,824 bytes.
terabyte, Tbyte, TB		A terabyte is 1,024 gigabytes or 1,099,511,627,776 bytes.

It is not uncommon today to hear some large data warehousing sites talk in terms of terabytes. In the next few years, we will probably hear of systems using storage in the tens and hundreds of terabytes.

Summary

This chapter defined and described various concepts with the intent of preparing you for the upcoming chapters. Be sure to look in the glossary for any other terms with which you are not entirely familiar.

Chapter

What Is a Well-Tuned System?

This chapter examines the attributes of a system that is performing well. A “well-tuned system” is a general term; as you will see in the next chapter, what is a good performing system for one site may not be the most optimal system for others.

Each system has its own performance criteria. In some cases, a system is tuned well if a strict response time criteria is met. In other situations, a system is tuned well if the throughput is optimized. In other cases, the system may be optimally tuned for backup and recovery. It is up to you to determine your performance criteria.

This chapter looks at some general cases and criteria for a well-tuned system. Later chapters in the book focus on specific problems and solutions that may be standing in the way of achieving a well-tuned system. It is important to look at the system as a whole and try to determine how it is balanced. This chapter is split into different sections for *client/server* computing, *terminal-based* (also known as *host-based*) computing, and *batch* computing. The end of the chapter lists a few exceptions to the general rules presented in the first parts of the chapter.

Client/Server Computing

Throughout the book are references to the *client* or *front-end machine*. This book defines the *client* to be the computer that requests information and presents this data to the user. In the case of a PC running Personal Oracle for Windows, the client and the server are the same machine. Similarly, a UNIX computer connected to terminals is also considered both the client and the server. The terminal does not present data to the user; rather, the UNIX system presents the data that the terminal merely displays.

In this book, I refer to a client as a computer—PC or otherwise—separate from the database server. Host-based computers are systems that contain the RDBMS and present the data to terminals.

Here's an example of a client/server application performing a transaction. The work is split into the following steps:

1. Load and initialize the application. This step includes loading the application code into memory and displaying the application screen(s).
2. Connect to the database server. This step can be accomplished with SQL*Net, ODBC, and so on.
3. Accept data from the user. Data can be submitted from the keyboard, mouse, barcode scanner, or some other means.
4. Process the data into the proper form to transmit to the server. This generally means forming the data into a structure that is passed to the database protocols along with any control information that must be added (such as the transaction type, amount of data being transmitted, and what stored procedures are to be executed).
5. Transmit the data to the server.
6. Wait for a response from the server.
7. Receive the data back from the server. In this step, some error checking is typically done to verify that the server has not sent back any error codes.
8. Format the data back into a form that the application code needs for its processing.
9. Bring up additional applications (optional). An application may invoke another application such as a spreadsheet or multimedia viewer.

10. Present the data to the user. This is done in whatever format the application requires (such as text, graphics, video, audio, and so on).

The following sections take each component involved in this process and examine what is optimal for it. The following sections look in turn at the client, the server, and the network used to connect the client and the server.

The Client or Front-End Machines

Typically, clients fall into one of several categories: those that perform purely presentation services, those that do some form of data manipulation, and those that may even have a local database for some static tables such as selection lists. If the client does data manipulation, the task may be as simple as sorting the data or as complex as joining the data from several different data sources.

For all three of these client models, it is important that the client has sufficient resources to perform its function. There should be no significant delays caused by presenting or manipulating the data. The performance the user sees is *end-to-end performance*. The time between when the last character is typed by the user and when the last bit of data is displayed is considered the *response time* (see Figure 3.1). The response time is the meter by which the user judges the performance of the system. In most cases, systems are tuned for this response time.

NOTE: In most cases, systems are tuned for response times. There are exceptions—such as when response time must be sacrificed to load large amounts of data within a certain time interval.

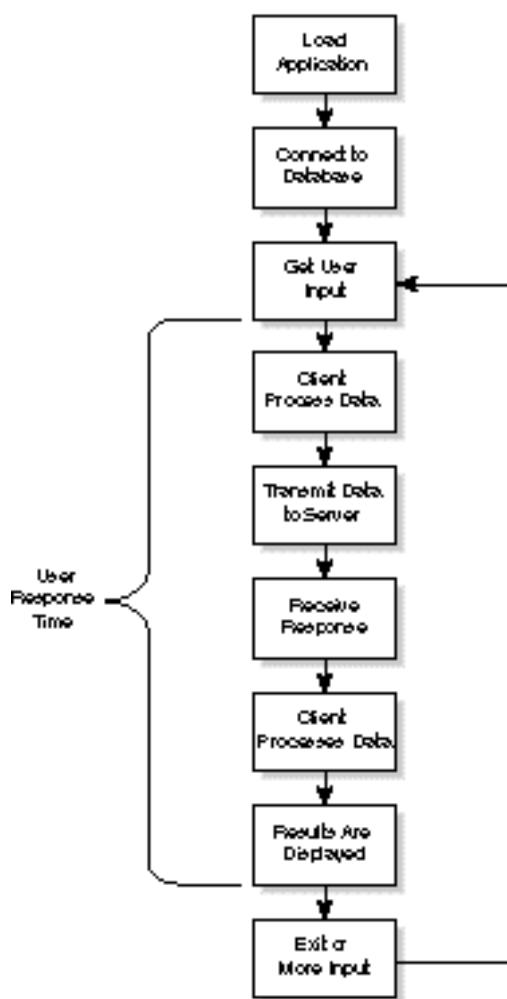
If the goal is optimizing response time, the criteria for a good-performing client is to add minimal delays to response time. It doesn't benefit you to have the fastest server available if a 10-second response time is caused by a 1-second turnaround at the server and 9 seconds displaying the data. The user experiences long response times and is not satisfied.

As you see in Chapter 4, “Tuning Methodology,” when you benchmark a system, it is important to keep in mind response times as well as system throughput. Most benchmark tests today have very tight response-time criteria.

The Server

It is the job of the server to take requests from the clients, process the data as quickly and efficiently as possible, and pass the data back to the user. If the client machine typically serves only one user, response time is the highest criteria. For the server, even though response time is important, throughput is typically the most important factor.

Figure 3.1
The user response time.



Because the server processes information for hundreds or perhaps thousands of clients, the needs of all the users are more important than the needs of any one particular user. It is the job of the server to process as many transactions as quickly as possible. This goal is accomplished by balancing the system to respond as quickly as possible to each request and at the same time to make sure that each request gets an equal share of the system's processing power.

A well-performing server provides good response times for each user's requests. In some cases, it may be desirable for some users to get a low priority (such as for a decision support task) while maintaining good throughput for OnLine Transaction Processing (OLTP) tasks, such as order entry and account inquiries.

In looking at the server's performance, you want the RDBMS processing to be a CPU-bound task. If disk access performance is a problem, you may be able to solve the problem by increasing the amount of memory in the system or by adding more disks (as you see later in this book). Because RDBMS processing is a CPU-intensive task, make sure that no other components in the system are holding back the CPUs.

TIP: Another important factor for all servers is their ability to perform backups in a reasonable time. Every system should have a good backup and recovery plan that is executed efficiently. If your system fails, it is only as good as its last backup.

Here are the attributes of a good-performing server:

- ◆ RDBMS processing is not bound by disk, memory, or network components
- ◆ Good throughput and response times are maintained
- ◆ The ability to perform backups within a specified time window

Of course, all installations are unique. You may have different criteria for what is important.

The Network

The *network* is defined as the logical and physical connections between the client and server machines. These connections include not only the wire, the network cards, and routers, but the TCP/IP, SPX, and SQL*Net components as well. The collection of all these components is necessary to connect clients and servers.

The network is typically described as one of those things that is ignored unless there is a problem with it—and I agree with this assessment. Network components should be configured and maintained with enough capacity that there are no tuning concerns with them. The design of the system should provide for enough capacity to handle peak as well as normal traffic; it should also make considerations for capacity growth.

If you exceed the capacity of your network, you will see significant performance degradation in both throughput and response time. It is a good idea to keep an eye on your network traffic and try to avoid problems before they occur.

Your network should also consider delays incurred through routing and bridging to other networks. These delays—although typically insignificant—can become a problem if they are not managed effectively. These factors should be considered during the design phase.

Client/Server Checklist

In a client/server system, each component is important and affects the performance of the entire system. Response time delays can occur in the client, the server, or the network. Each component should be configured and tuned for both response time and throughput.

Here is a short checklist of items that may indicate whether you have a well-tuned system:

- ◆ Do you have good response times? If not, is the problem in the client, the server, or the network?
- ◆ Can the server effectively service the number of clients required?
- ◆ Are the users satisfied with the performance of the system?
- ◆ Are time-critical tasks such as backup and loading being accomplished within the required time intervals?
- ◆ Is the system disk bound?
- ◆ Is the system network bound?

This book will help you determine whether any of these items are problems; if they are, you will learn how to fix them.

Host-Based Computing

Until recently, almost all applications were host based. Today, most applications still use terminals or PC terminal emulators as the display devices. It is not feasible or cost effective to migrate many applications to client/server systems just to make them client/server applications. As with client/server applications, you must ensure that both the front-end and the RDBMS are both working effectively if you are to have a good-performing system.

The Front-End Application

Unlike the client/server configuration, the terminal-based application does not have to worry about effectively sizing the client computer. Terminal applications more strongly emphasize tuning the application code for efficiency because they don't usually have excess CPU power and memory resources (as is the case with some clients). Terminal-based applications must also consider that they share CPU processing and memory with other users on the system. Any waste in CPU cycles and memory usage is compounded by the number of users sharing the application.

As with client/server applications, make sure that the terminal-based application does not spend too much time formatting and presenting the data. Even in terminal-based systems, inefficient coding can add seconds to the time it takes to present the data. Consider offloading some of the terminal processing to other devices such as a terminal server or a smart terminal controller. Keep in mind that every inefficient statement you optimize has an effect that is multiplied by the number of users on the system.

The Database

Typically, host-based configurations—as well as client/server configurations—should be tuned for maximum throughput with a consideration for response times.

In terminal-based configurations, as in client/server configurations, it is the job of the RDBMS engine to process the data as quickly and efficiently as possible and to pass this data back to the user. Although the computer is responsible for both database processing and presentation services, throughput is typically the most important factor. Throughput is important because there are many users to service; reducing response time for a particular user may not serve the needs of the entire user community effectively.

A well-performing system provides both good response times for the users and good throughput for the entire user community. You must ensure that a single user process does not excessively burden the system at the expense of other users. System administrators may choose to tune the system for OLTP users or for other tasks such as backup/restore, loading of data, or decision support tasks.

In looking at the system's performance, you want the RDBMS processing and the screen handling to be CPU-bound tasks. If disk throughput is a problem, that can easily be solved. Because RDBMS processing is a CPU-intensive task, make sure that no other components in the system are holding back the CPUs.

Here are some of the attributes of a well-performing host-based system:

- ◆ RDBMS and screen-handling processes are not bound by disk, memory, or network or terminal-handling components
- ◆ Good throughput and response times are maintained
- ◆ Effective backup of the system is possible within a specified time interval

Of course, all installations are unique; you may have different criteria for what is important.

NOTE: If the screen-handling processes are too intense for one system to handle, you may be able to move the screen handling off the database system by separating the application into two parts: one for the RDBMS functions and one for the screen-handling functions. By separating the functions, you can then offload the screen-handling function to a client system. Remember that a *client* is defined in this book as a system that handles presentation services. A client can be a multiuser machine that serves terminals.

Terminal-Based Checklist

As with client/server computing, it is important with terminal or host-based computing to look at the efficiency of the entire system. Any single component can drag down the system performance. Look at each component as an individual contributor to the entire system. By minimizing resources used in the application code, the effect of the reduction is multiplied by the number of users simultaneously using this code.

Here is another checklist of items that may indicate whether you have a well-tuned system:

- ◆ Do you have good response times? If not, is the problem in the application code, in the database, or in the operating system?
- ◆ Can the system handle the required number of users effectively or do response times drastically increase with the numbers of users?
- ◆ Is the user community satisfied with system performance?
- ◆ Are time-critical tasks such as backup and loading accomplished within the required time interval?
- ◆ Is the system disk bound?

Later chapters examine how to determine whether some of these things are problems and how to fix them if they are.

Batch Computing

Batch computing refers to large jobs that take a significant amount of time and that typically aren't accessed in an interactive manner. Response times can be minutes or even hours. Depending on the application, there may be large amounts or very small amounts of data returned. Usually, batch jobs consist of two phases: the batch loading and the batch processing. In both phases, processing time should be optimized.

What determines a well-tuned batch-processing system? The system should be optimized for load time as well as for batch processing. If the system is dedicated to batch processing, OLTP queries may see significant increases in response times.

The system should load and process data in the required time periods. The system should be CPU bound. The system should be able to transfer large amounts of data effectively over the network or through removable storage as specified by the application.

If large amounts of data are to be moved across the network, you should ensure that the network does not become a bottleneck. If necessary, install faster network hardware or segment the network to have a dedicated connection between the batch system and the loading system. If this is not possible, schedule the activity at a time when it least affects other users on the network. You may also be able to break the data into pieces for transfer or employ alternative methods such as tape.

Because the job in batch-processing systems is to move large amounts of data in, process that data, and move the results out, backup and recovery may not be important considerations. But remember that if all the input data is saved, you can always rerun the job to achieve the same results.

A well-tuned system can effectively handle these bursts of activity. A major difference between an OLTP system and a batch system is that—usually—the OLTP system has a constant workload; the batch system has bursts of activity.

Batch-Processing Checklist

Batch-processing systems are different from OLTP systems in that they must be able to handle the loading of large volumes of data and process large amounts of data in bursts. Typically, there are only certain time windows in which this processing can take place. Concern for response time is replaced by concern for processing times that can take minutes or hours to complete.

In a batch-processing system, the checklist for a well-tuned system is much shorter:

- ◆ Are you able to process the data in the required time? This includes loading, processing, and offloading the data.
- ◆ Is the system disk bound?

This book examines how to determine whether some of these things are problems and how to fix them if they are.

Exceptions

I always find it interesting to look at some of the cases that are exceptions to the typical way most systems operate. The following sections describe some of the systems that handle things a little differently.

Multimedia Systems

Multimedia systems have slightly different criteria about what a well-tuned system is. In a data-processing environment, slight interruptions in response times are hardly noticed. In video and audio processing, a slight interruption in the constant stream of data is very noticeable. Therefore, in a multimedia environment, response times and ensuring a constant stream of data are the highest priorities.

Shipping Systems

Some applications in the shipping industry have such transient data that backups may not be significant. Consider a database that contains tracking information for packages that are picked up, moved around the country, and then delivered. In the time it takes to do a backup of this system, the data for a particular package may no longer be in the database. If a hardware failure occurs, by the time the system is back online and data is restored, perhaps none of the items in the database are of any use. For such systems, it may be more important to configure for fault tolerance. The system can be tuned for maximum throughput with less emphasis on tuning for backup and recovery.

Of course, account information must be backed up in any event. This type of application is a candidate for a fault-tolerant system such as the Oracle Parallel Server or the Oracle Standby Database feature available in Oracle version 7.3.

Summary

This chapter looked at several different types of systems, each of which has different criteria for being well tuned. A system that is well tuned for client/server applications may not be the best-tuned system for batch processing. However, there is a commonality: in each case, you want to make sure that individual components such as disks and networks are not slowing down the system processing.

Another important factor in a well-tuned system is the ability to handle peak events. If the load on the system is increased—caused by an unusually high number of users or an event such as a system backup or a checkpoint—the system should still be able to process transactions at nearly the same rate as during the typical system load.

You will have slightly different criteria for what you consider a well-tuned system depending on your needs. The following chapters discuss how to tune the system to get the results you want.

Chapter **4**

Tuning Methodology

This chapter covers tuning methodology—the manner in which you analytically determine what it is you want to tune and for what ends. If you set out to tune or optimize a system without any goals, there is nothing to measure success against. The first part of the chapter looks at some of the tuning goals you might have; the second part of the chapter is an introduction to a tuning methodology that may help you in your tuning efforts.

Goals

It is important to set tuning goals for your system. Each system has different characteristics depending on the needs of the users. You must determine why you are tuning a particular system. The desired goals often influence the kind of changes made and whether those changes are successful or not. People tune their systems for different reasons; the following sections examine a few of them.

Throughput

Throughput is defined as the amount of work done divided by the time it takes to do that work. Work to be done is usually defined in terms of *transactions*. Therefore, the throughput of the system is defined as follows:

$$\text{Throughput} = \# \text{Transactions} / \text{Time}$$

The *time* in this equation is usually some measurement interval during which a large amount of work is done. Consider, for example, a company that takes reservations. If the reservation office is open 8 hours a day and the number of tickets sold for an event must be 25,000 for first day of sales, the required throughput for this system must meet or exceed 3,125 transactions per hour:

$$\begin{aligned}\text{Throughput} &= (25,000 \text{ Transactions} / 8 \text{ hours}) = 3,125 \text{ Transactions per hour} \\ &\text{or } 52.1 \text{ Transactions per minute}\end{aligned}$$

Depending on the complexity of the transaction and the granularity of your measurements, you can express throughput in transactions per hour, transactions per minute, or transactions per second. For some large decision support system (DSS) applications, you may even measure throughput in transactions per day.

For your particular configuration, the most important tuning goal may be system throughput. In some environments such as OnLine Transaction Processing (OLTP), it may be vital to achieve the highest throughput possible, thus getting the most possible work through the system in the shortest possible time.

To achieve the highest possible throughput, you may have to make sacrifices in some areas such as response time and perhaps fault tolerance and recovery time.

Response Time

Response time is the time from when you press the last key for an input form until all the data has been displayed on the display device. Response time is essentially the time the end user spends waiting for the behind-the-scenes processing of the job.

Although some installations require the highest throughput possible, others may have strict response time criteria. In such an environment, slow response times usually mean customers waiting on a phone line—thus keeping other customers on hold.

For installations that have strict response time criteria, you may have to tune the system differently. It may be necessary to run the system at a much lower throughput rate than it can handle or to delay batch jobs until off hours.

Connectivity

Connectivity (the ability to support connections to other systems or clients) may be an important factor in configuring and tuning your system. You may have to configure and tune your system to support large numbers of users in an effective manner. Additionally, you may have to incorporate sufficient capacity to support additional users. Such a system might be used for a business that requires additional employees at various peak times during the year.

To tune for large numbers of connections, memory can be a special concern. You must carefully plan for the memory requirements of these users and make sure that you don't exceed available resources during peak periods. This may mean that you have to configure extra memory into the system that is on stand-by for user connections. OS resources associated with network and user connectivity must be closely monitored.

You can determine the resources associated with these users experimentally. Try running the application with a specific number of users and monitor system resource usage. Increase the number of users and monitor the system again. Comparing the results will give you a fairly good idea of the amount of resources associated with each user.

TIP: When performing these types of tests, it is critical to take good notes. By logging configuration changes and their results, you can get a good idea of how those changes affect the system.

Fault Tolerance

For some installations, it is of the utmost importance that *fault tolerance* be employed in every aspect of the system. Any down time in such a system may be devastating. A site with high fault tolerance requirements may require frequent checkpoints and frequent backups.

Several tuning considerations come into play when fault tolerance is the highest priority. The disk subsystem should use some type of hardware Redundant Array of Inexpensive Disks (RAID) to protect against disk failures. The memory should be protected with advanced ECC memory. When uptime is critical, you may want to consider a redundant system, which you can use if the primary system fails.

Load Time

Some systems have a requirement that a certain amount of data must be loaded each night to be available for the next day's processing. Usually, the load time is limited and it is essential that a specific amount of data be loaded within a certain time. Although the load time requirement can be met, doing so may be costly in terms of additional hardware that may be required.

In this situation, you may have to tune the I/O subsystem for load time. However, configuring for load time may affect the general performance of the system. By tuning for both load time and run time, you can minimize the affect.

NOTE: It is important to realize that your tuning goals and my tuning goals may not be the same. Depending on your installation, there may be drastic differences in configuration. Each site is different and serves a different function; each site must be analyzed individually.

Tuning Methodology

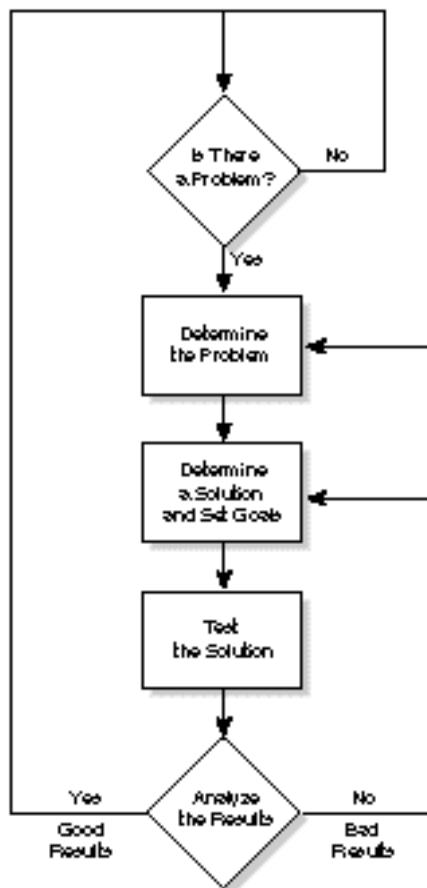
This section explains why it is important to follow a structured, goal-oriented methodology in your tuning efforts if you want to achieve optimal results. The methodology I use involves these five steps as shown in Figure 4.1.

1. **Analyze the system.** Decide if there is a problem; if there is a problem, what is it?
2. **Determine the problem.** What do you think is causing the problem? Why?
3. **Determine a solution and set goals.** Decide what you want to try and what you think will result from your changes. What is it you want to accomplish? Do you want more throughput? faster response times? what?
4. **Test the solution.** Try it. See what happens.
5. **Analyze the results.** Did the solution meet the goals? If not, you may have to go back to step 1, 2, or 3.

The following sections describe each of these steps individually and explain why each step is important to the final outcome. Sometimes, there are situations for which you don't have an answer to the problem or perhaps you don't even know what the problem is. In the long run, however, a little analysis is worth more than a lot of trial and error.

Figure 4.1

The tuning methodology flowchart.



Examine the Problem

The first phase in tuning the system should always be the determination phase. First, see whether you even have a problem. You may just be at the limits of the configuration. To make sure, examine the system, look for problems, and make this determination logically.

The determination phase consists of examining the system as it currently is, looking for problems or bottlenecks, and identifying “hot spots.” Keep in mind that performance problems may not always be obvious. Here is a partial list of some areas you should examine:

- ◆ **Application code.** The problem may consist of an application that executes excessive code or performs table scans when they are not necessary.

- ◆ **Oracle database engine.** The problem may be an improperly sized System Global Area (SGA) or some other tuning parameters.
- ◆ **Operating System parameters.** The OS may not be configured properly and may be starving Oracle of needed resources.
- ◆ **Hardware configuration.** The layout of the database may not be efficient and may be causing a disk bottleneck.
- ◆ **Network.** The network may be overloaded and causing excessive collisions and delays.

I prefer to categorize performance issues into the following three classes:

- ◆ **It's broken.** Performance is severely handicapped because of a configuration error or an incorrect parameter in the OS or RDBMS. I consider problems that fall into this category to cause a performance swing of 50 percent or more. Problems in this category are usually oversights during the system build (such as incorrectly building an index or forgetting to enable Asynchronous I/O).
- ◆ **It's not optimized.** Performance is slightly degraded because of a small miscalculation in parameters or because system capacity is slightly exceeded. These types of problems are usually easily solved by fine tuning the configurations.
- ◆ **Not a problem.** Don't forget that sometimes there isn't a problem—you have just reached the capacity of the system. This situation can easily be solved by upgrading or adding more capacity. Not all problems can be solved with tuning.

For the first class of problem, you may have to take drastic action. Tuning may solve the second class of problem. For the third class of problem, you need take no action.

Here are some questions to ask yourself when examining the system and searching for problems:

- ◆ Are you getting complaints from the user community?
- ◆ Are some operations taking much longer than the same operations did in the past?
- ◆ Is CPU usage low but I/O usage high?
- ◆ Are you seeing excessive response times?
- ◆ Does the system seem sluggish?

If you answer *yes* to any of these questions, you may be seeing some performance problems that can be fixed. In some cases, you have reached the limitations of your hardware and should add an additional CPU or perhaps some more disks.

If you answer *no* to all these questions, your system may be running very well. However, you should still periodically monitor the performance of the system to avoid future problems.

Determine the Problem

Once you have decided that there is some sort of problem in the system, make a careful analysis to determine the cause of the problem.

At first glance, it is not always obvious what the problem is. You may see excessive disk I/O on a certain table that may lead you to believe you have an I/O problem when, in fact, you have a cache-hit problem or an index problem that is causing excessive I/Os.

Take a few minutes to think about the problem. Decide what area you believe the problem to be in. Thinking through the problem can help you avoid a long trial-and-error period in which you randomly try different things without positive results.

At this stage, note-taking is a must. By writing down your ideas and thoughts, you can better organize them. If you have to go back and rethink the problem, your notes will be very useful.

Look at the chief complaint and try to understand how that component works and what can be affecting it. Suppose that you are having a client/server response time problem. The following sidebar analyzes this problem.

Response Time Example

The user response time is measured from when the data is input and submitted until the results are displayed on the user's display device. To analyze a response time problem, start by considering the components. The problem can be broken into three major areas:

- ◆ **Client.** The problem could be with the client application code.
- ◆ **Server.** The problem could be with the database engine.
- ◆ **Network.** Some problem in the network could be delaying processing.

You can use analysis tools to help determine the area to which you can isolate the response time problem. By putting debug statements into the application code, you can determine whether the delays are part of the client component. By using the EXPLAIN PLAN command, you may see that you are not taking advantage of an index and are causing excessive table scans.

I have heard of several examples in client/server systems in which the end user complains about the performance of the server: it is causing 10-second or greater response times. After analysis, it was determined that the database was giving subsecond response times but was spending over 9 seconds in the GUI (Graphical User Interface).

Be open minded and look at every component of the system as a potential for bottlenecks. You may be surprised to find that the problem is something simple and in an area you would not normally think would cause problems.

Here are some questions to ask yourself when trying to determine the cause of the performance problem:

- ◆ What are the effects of the problem: response times? throughput? what?
- ◆ Does every user see the same problem or does the problem affect just a few users? If so, what are those few users doing differently?
- ◆ What does the system monitor tell you? Are you 100-percent CPU bound? What about I/Os: are you exceeding I/O limits?
- ◆ Is only a certain type of transaction a problem? What about other transaction types? Are they okay?
- ◆ Is it a client problem or a server problem?

By asking yourself these questions and making up your implementation-specific questions, you will get more insight into how the application operates and where the potential bottlenecks are.

Determine the Solution and Set Goals

Regardless of the type of problem you have, you must determine a course of action. In many cases, determining the problem has already pointed you to a solution. In other cases, more analysis is necessary. Once the problem has been determined, you must set some goals about what you want the solution to accomplish. When you set specific goals, not only can you measure success, but specific goals may sometimes point you to specific solutions.

The solution may consist of changing a parameter in the OS or in Oracle; the solution may consist of a rewrite of a section of application code. With packaged applications, you may have to tune the OS and database to overcome a problem in the application.

The determination of a solution should include some expectation of what results you expect to see. Changing a parameter without at least knowing what it is supposed to do may provide the desired results or may instead mask the true problem.

Goal Setting

Once you determine the problem (or at least have a pretty good idea about the area causing the problem), you should set some goals. Your ultimate goal is always optimal performance but here I want you to think more specifically.

Your goals should be specific and achievable. Here are some examples of realistic and achievable goals:

- ◆ Increase the cache-hit ratio in the buffer cache.
- ◆ Reduce the number of I/Os.
- ◆ Increase the number of connections supported.

Goals such as these may or may not be achievable in your system but at least they give you an area on which to focus your tuning efforts. Unachievable goals always end in disappointment. Although we all have the goal to have the best-performing system possible, smaller, more specific goals can help you achieve the larger goal more effectively.

Here are some questions to ask yourself when you are determining the solution and setting result goals:

- ◆ What will this solution do? What do you expect the result of your changes to be?
- ◆ How will your changes affect performance? For example, does a better cache-hit rate means less I/Os?
- ◆ Will this solution directly or indirectly affect the problem?
- ◆ How will your idea of the problem change if this solution doesn't fix the problem?

These are just a few examples of the kind of things to think about when determining the solution and setting goals.

Test the Solution

Once you put the solution in place, you must test the solution to see whether you have achieved the desired results. Although you may be able to test the results in a test configuration by setting up an emulated load, you may have no choice but to put the system into the production environment for analysis.

During the testing stage, it is also very important to take notes. These notes can be extremely valuable in the future when similar problems arise and you are looking for solutions.

CAUTION: I don't recommend putting test configurations into production unless you completely understand the changes that have been made and have assessed the risks.

In most cases, you will have the test programs and user-emulation facilities you used in the development phase. You can use these to test the new configuration against already-known performance results. Refer to Chapter 5, “Benchmarking,” for more information on this topic.

Analyze the Results

The final step is to analyze the results of any changes you have made. If the desired results have not been achieved, go back to the analysis stage to determine whether more tuning is necessary. You should always ask the question *why?* It is possible that the problem has been incorrectly identified or that you have not chosen the correct solution for that problem.

Here are some questions to ask yourself during the analysis phase:

- ◆ Did the change have the desired results?
- ◆ Did the overall system performance change?
- ◆ Do you now think that the problem is different from what you first thought?
- ◆ Do you think more changes are necessary?

These are some examples of the types of questions you should ask yourself to determine whether the problem has been solved—or even affected—by the changes. You may have to go back and reanalyze the problem and try different things. Maybe you did not understand the problem correctly and have to add some additional debug code to the application to track down the problem.

This five-step methodology leads to an analytical, logical way of viewing performance data. Using this—or any—methodology should lead you to the proper solution.

Summary

Develop a scientific approach to tuning and follow it. I prefer this five-step method:

1. Analyze the problem. Determine whether something is broken, whether it is not tuned well, or whether you are at the limits of the system.
2. Determine what, specifically, you think the problem is. Don't look just at the superficial; try to determine the root of the problem or bottleneck.
3. Develop a solution. Decide what you want to change and what you believe the result of these changes will be. Set realistic goals. Set achievable but aggressive goals. Don't expect enormous results from fine tuning.
4. Test the solution. You have to test the solution to determine whether or not the change is effective.
5. Analyze the results. See whether you achieved the desired results. If not, what went wrong and why?

This methodology may or may not work for you. If you prefer a different system, by all means use what works best for you. But use some sort of system or methodology for tuning. When following a tuning methodology like this, it is important that you take careful notes. If several iterations are done without careful notes, you may not be able to reproduce the results.

I hope that this system works for you and helps you achieve great results. In any case, you now know the importance of having a systematic approach that includes good documentation.

Chapter 5

Benchmarking

A benchmark is a test used by computer hardware, software, and application vendors to test the performance of a system. Benchmarks are designed so that the same test can be executed on a variety of hardware and software platforms. Because the same tests can be run on a variety of systems, the results can be compared to give a relative indication of performance.

Because benchmarks usually test a specific function, they can give only an indication of the system's performance in that area. You may find a system that performs extremely well on an OLTP benchmark but does not perform well in a decision support environment.

There are essentially two categories of benchmarks: standard and custom. The industry standard benchmark is typically run by the computer hardware or software manufacturer; the custom benchmark is usually used by the end user or integrator.

The *industry standard benchmark* allows computer hardware and software vendors to compare the performance of their systems against other vendors' systems. These benchmarks have been designed by the industry as a whole to provide a fair and level playing field. This chapter looks specifically at the TPC (Transaction Processing Performance Council; in its early days, the council shortened its acronym from TPPC to TPC), which is dedicated solely to database benchmarks.

The *custom benchmark* is used extensively throughout the industry; it is designed by the user to compare systems with a test that closely mirrors the systems' environments. Custom benchmarks are also widely used in the computer industry to allow the manufacturers to test specific components and make improvements or overcome deficiencies.

Custom benchmarks are also used by publications to test certain configurations of hardware or software. These custom benchmarks usually have strict configuration requirements such as hardware limitations, OS tuning constraints, and RRDBMS layout so that they can provide a comparison of a specific subcomponent of the system.

Introduction to Benchmarking

A benchmark is designed to test a functionality of a system and to report its performance. Some benchmarks are designed with general functionality in mind; others are quite specific. Database benchmarks are typically divided by transaction types. By far, the majority of database benchmarks are OnLine Transaction Processing (OLTP) with a few decision support benchmarks now available. Of course, custom benchmarks span the entire range of database transaction types.

This chapter first looks at the industry standard benchmarks available from the TPC. Then it looks at how you can design your own benchmark. The industry standard benchmarks are useful for giving you a rough estimate of how a particular system compares with other systems on the market. This comparison may allow you to shorten the list of systems you need to test before making your final decision. Custom benchmarks are great for giving you a real feeling about how a particular system will perform in your specific environment.

Industry Standard Benchmarks

Industry standard benchmarks are designed and maintained by organizations made up of computer hardware, operating system, and database-related companies that have a stake in seeing that the tests are fair. In the case of the TPC, these companies also police each other to make sure that fairness is maintained.

Because the organizations are made up of so many competing companies, the test results are usually seen as fair and comparable. Once the benchmark designs are completed and companies begin to run the benchmarks, competition sometimes becomes quite fierce. Each company wants to be seen as publishing the fastest and most price-competitive benchmark.

A phenomenon that is quite apparent in the database performance arena is that systems previously thought of as low-end are increasing in performance at an incredible rate. PC servers, for example, are achieving database performance that had been reserved for mainframes just a few short years ago.

The following sections take a look at the TPC and some of its benchmarks and give some information about where you can get the results of these benchmarks and how to interpret them.

The Transaction Processing Performance Council (TPC)

The Transaction Processing Performance Council (TPC) was founded in 1988 by eight computer hardware and software companies. At that time, there was no industry standard benchmark on the market; companies had a difficult time trying to compare themselves to other companies.

The TPC was founded with the goal of defining transaction processing benchmarks that provided objective and verifiable performance data. The objectives defined by the original eight companies are still followed today. These objectives are to provide industry standard benchmarks with the following qualities:

- ◆ Complete and thorough specifications that allow anyone to implement the benchmarks regardless of system architecture.
- ◆ Objective and verifiable results that do not handicap any particular architecture.
- ◆ Full disclosure of the system configuration, cost, and benchmarking methodology. This allows results to be independently reproduced.
- ◆ A fair means of comparing one system against another.

These qualities went into the first benchmark developed by the TPC and are still present in all the benchmarks developed today. In the years since the TPC was originally founded, it has grown from the original 8 members to over 40 of the leading computer industry companies. As of January 1, 1996, the TPC membership consists of the following companies:

Amdahl	Mitsubishi Electric Corp.
AT&T	Motorola
Australian Dept. Admin. Svc.	NEC Systems Laboratory
Bull	Mikkei Business Publications
Compaq Computers	Novell
Convex Computer Corp.	OKI Electric Industry
Cray Research	Olivetti S.P.A
Data General Corp.	Oracle Corp.
Digital Equipment Corp.	Pyramid Technology
EDS	Samsung
EMC Corp.	SCO
Encore Computer Corp.	Sequent Computer
Fujitsu/ICL	Siemens Nixdorf Information
Hewlett-Packard	Silicon Graphics
Hitachi SW	Software AG
IBM Corp.	Sony Corp.
IDEAS International	Stratus Computer
Informix Software	Sun Microsystems
Intel Corp.	Sybase
Intergraph	Tandem Computers
ITOM International	Toshiba Corp.
Microsoft Corp.	Tricord Systems Inc.
	Unisys Corp.

The TPC is made up of the general membership, in which each member company has one vote. From the general membership, a steering committee and Technical Advisory Board are elected.

The steering committee is responsible for setting the overall direction of the council and providing recommendations to the council. It is also charged with the administrative and support activities of the council.

The Technical Advisory Board (TAB) investigates issues involving compliance to TPC specifications and makes recommendations to the council on these matters. The TAB is also responsible for recommending changes to the benchmark specifications for clarity.

Any change in a specification or compliance issue decided by the TAB must be brought to the general council for decision. Every issue in the TPC requires a 2/3 majority for passage. The steering committee and TAB are merely filters; all significant decisions are made by the general council.

The TPC has gone from the original Credit/Debit benchmark to seven benchmarks (by mid-1996). These benchmarks include Decision Support, Client/Server, and Enterprise benchmarks. Each of these benchmark specifications is discussed in this chapter.

TPC Rules and Regulations

To publish a TPC benchmark result, the sponsoring company is required to follow a set of rules and regulations. These rules are designed to provide a fair playing field for all participants. Here are some of the significant rules:

- ◆ Only results that have been published can be advertised as TPC results.
- ◆ Once results are submitted to the TPC, there is a 60-day window in which the results can be challenged by other members of the council on the grounds that the benchmark does not comply with the specification. If the challenge is upheld, the results must be removed.
- ◆ Estimated results cannot be advertised.
- ◆ Comparisons cannot be made between different major releases of the TPC specifications (for example, you cannot compare TPC-C 3.x results with TPC-C 2.x results).
- ◆ Since January 1, 1994, all TPC benchmarks must be independently audited by a certified TPC auditor.
- ◆ Full disclosure reports must be submitted with all TPC results.
- ◆ Published results must include all primary metrics. For most TPC benchmarks, this means both performance rates and price/performance but may include more.
- ◆ TPC benchmarks support ACID properties (Atomicity, Consistency, Isolation, and Durability, as described in the following section).
- ◆ The availability date of nonshipping components must be disclosed.

These rules have been approved by the council and must be followed by all members. If these rules are violated, the council has steps it can take. It is very rare that any of the rules are violated—and then usually by mistake.

ACID Properties

The ACID (Atomicity, Consistency, Isolation, Durability) properties are designed to show that the system operates in a manner consistent with systems deployed in user installations. ACID properties are the main regulating factor that keeps benchmarks from being published in a mode inconsistent with typical operation. Consider a case in which the system might run faster if logging were disabled. Although the performance increases, the test is inconsistent with typical operation because disabling logging prohibits any recovery in the event of a system failure.

The ACID properties are as follows:

- ◆ **Atomicity.** The system must guarantee that all operations are atomic. This means that a transaction must complete all individual operations on the data or guarantee that partial operations have no effect on the data.

- ◆ **Consistency.** Consistency requires that any execution of a transaction leaves the database in a consistent state, assuming that the database is in a consistent state to begin with. This property guarantees that the books are balanced. An example of this property in the TPC-B benchmark (described later in this chapter) requires that the sum of all the account balances for a teller must match the teller balance, and that the sum of all teller balances in a branch must match the branch balance.
- ◆ **Isolation.** Isolation requires that operations of concurrent transactions must yield results indistinguishable from the results obtained when you execute the transactions serially in some order. This property is commonly known as *serializability*. Furthermore, this property must be guaranteed with respect to arbitrary transactions as well as other benchmark transactions. Repeated reads of the same records within any committed transactions must also be guaranteed to return identical data when run concurrently with any mix of arbitrary transactions.
- ◆ **Durability.** The durability property requires that the system must demonstrate that all transactions returned as completed have in fact been committed in spite of any system failure that may have occurred. To demonstrate this, a system crash test must be run and it must be shown that no completed transactions have been lost.

These properties guarantee the integrity of the system and add credibility to the benchmark results. Although these tests are quite time consuming, they are required for a certified result.

Requirements like these give a lot of credibility to the TPC and to the vendors that run the tests. To pass these ACID tests, the database platform must prove it is a stable, robust system capable of handling your business.

Results

The ultimate outcome of running a TPC benchmark is to publish the performance of the system. Other results are published as well. For a TPC benchmark to be valid, a Full Disclosure Report (FDR) must be submitted. The FDR includes the Executive Summary, which has highlights of the FDR and usually includes the following information:

- ◆ **The benchmark sponsor(s).** These are the companies that actually did the work for the benchmark. Typically, the primary sponsor is the hardware vendor and may additionally include the OS and RDBMS vendors.
- ◆ **The system performance and price/performance.** These are the primary metrics used to compare system performance.
- ◆ **The system configuration.** This includes the system model number, number and type of CPUs, number of controllers and disk drives, network configuration, and so on.
- ◆ **A price list of all the components in the system.** This is a line-item list of all the components used, including part numbers and prices. The list also includes the support costs, which are required to be disclosed.

- ◆ **The response times of specific transactions.** Depending on the benchmark, this may include keystroke times, transaction response times, and think times.
- ◆ **Transaction mix.** Depending on the benchmark, the mix that was achieved is disclosed.
- ◆ **The configuration size.** Depending on the benchmark, this can be the number of users simulated, the number of warehouses used, and so on.

As you can see, the Executive Summary is valuable in providing a quick reference to the benchmark and the performance of the system. The Executive Summary is typically two to three pages in length and is available electronically from the TPC Web site (at this URL: <http://www.tpc.org/>).

In addition to the Executive Summary, the FDR includes the following information about the benchmark:

- ◆ **Configuration details.** Differences between the simulated configuration and the tested configuration are disclosed.
- ◆ **Response time graphs.** Graphs of the response times for each transaction type are provided.
- ◆ **Attestation letter.** A letter from the independent auditor attesting to the fact that the benchmark met all the rules of the specification.
- ◆ **Price quotations.** In some cases, a price quotation must be provided to verify the cost of the system.
- ◆ **Tuning parameters.** All OS and RDBMS tuning parameters must be disclosed. This information allows the benchmark to be reproduced by a competitor to verify the performance results.
- ◆ **Source code.** All application code must be disclosed. A rule was passed in the TPC in 1994 that allows all source code published in a benchmark to be considered public domain software within the TPC. This allows other vendors to copy the application code to be used in their benchmarks.

The FDR is quite detailed and provides a great deal of information about the test that was run. Usually, the vendor that ran the benchmark is happy to send you a copy of the FDR if you are interested.

Of course, the content of the FDRs and Executive Summaries varies depending on the benchmark that was run. The following section looks at the benchmarks that have come out of the TPC.

Benchmarks

This section describes what is involved in publishing a TPC benchmark. To publish a benchmark, the following steps are typically involved:

1. **TPC application.** The TPC benchmarks are only specifications for a workload. Because of the diversity of OS and RDBMS products, it is necessary to develop specific code for your particular RDBMS and OS. This code must be completely compliant with the workload defined in the specification. It is up to the vendor running the test to develop this application.
2. **Tune.** Once you have working benchmark code, the tuning stage begins. It can take days, weeks, or even months to achieve optimal performance. There may be additional tuning involved to meet the response time criteria or other part of the specification.
3. **Simulate.** You must either design and build your own keystroke-generator product or purchase one of the commercially available products. In several of the TPC benchmarks, you must actually press keys for the application and verify that the proper responses are sent back from the application.
4. **Audit.** An independent auditor must come on-site at your expense to verify compliance with the specification. The auditor also verifies that the system passes all the ACID tests. The audit can take several days or weeks.
5. **Full Disclosure Report (FDR).** A full disclosure report must be written to specify the performance rates achieved, system cost, and adherence to specification; the report must also include all source code used in your application.
6. **Marketing.** Only after the FDR has reached the TPC can you announce the TPC benchmark results you have achieved.

As you can imagine, TPC benchmarks can be quite time consuming and expensive. It is not only expensive to audit and run the test but the amount of hardware necessary to achieve the results can be quite large.

Although the benchmarks are expensive and difficult to run, they are seen as a great marketing tool. Each year since the TPC was founded, there has been an increase in the publication of benchmark results. As more and more benchmarks become available, I anticipate that the publication of results will continue to increase.

The year 1995 marked a milestone for the TPC with the retirement of its first two benchmarks: the TPC-A and TPC-B. The TPC is left with five active benchmarks: TPC-C, TPC-D, TPC-E, TPC-C/S, and TPC-Server benchmarks, each of which is described in the following sections.

Although the TPC-A and TPC-B benchmarks are now obsolete, you may have results from these benchmarks or may hear them mentioned. Because they are still referred to in the industry, they are included in the discussion of the TPC benchmarks that follows. Even though the TPC-A and TPC-B benchmarks have been officially retired and no more benchmarks can be *published*, these benchmarks are still *used* extensively within the computer industry. The TPC-B benchmark is an excellent load test that stresses several vital subsystems.

TPC-A

The TPC-A benchmark was adopted in October 1989 as the first benchmark published by the TPC. Formerly known as the Credit/Debit benchmark, the TPC-A measures performance in an update-intensive environment typical of OLTP applications.

The TPC-A benchmark simulates a banking system. Information is stored for each account, teller, and branch in this system. During a transaction, an account is either credited or debited. The corresponding teller and branch must be updated to reflect this transaction. This benchmark is characterized by the following elements:

- ◆ Multiple online terminal sessions
- ◆ Significant disk input/output
- ◆ Moderate system and application execution time
- ◆ Transaction integrity (ACID properties)

The TPC-A benchmark employs only one update-intensive transaction type to load the system. Although this transaction type does reflect some of the workload generated by OLTP transactions, it does not reflect the entire range of OLTP environments. This lack of variance in transaction types has contributed to the obsolescence of this benchmark—but having a simple, repeatable unit of work is still useful for exercising key system components. The TPC-A became obsolete on June 6, 1995. From this date on, no more TPC-A results can be published. As of December 6, 1995, all TPC-A results were removed from the official TPC results list.

The TPC-A benchmark is designed to show entire system throughput; as a result, “terminal to terminal” performance is measured. An actual terminal device is simulated and the response time incurred in the transaction is measured. The TPC-A specification requires that 90 percent of all transactions must have a response time under 2 seconds. The number of terminals is not fixed but is a factor of the transaction rate achieved. The specification calls for 10 emulated users per tps (transactions per second) reported.

The result of the front-end processing requires that additional machines be used in the benchmark to offload the work of the data input handling. Typically, a Transaction Monitor (TM) is used to multiplex these connections.

The TPC-A benchmark can be run in a wide area network (WAN) or local area network (LAN) configuration (the performance of the two modes cannot be compared). The Full Disclosure Report must report the cost of the system including 5 years of maintenance.

The TPC-A benchmark is scaled based on the performance reported: the larger the result to be published, the larger the size of the database. A 50 tpsA database has a size of 0.5GB; a 500 tpsA database has a size of 5GB. This configuration must also include enough disk space to provide for 90 days of online disk storage at the rate measured. This requirement significantly increases the size of the system being priced.

The metrics used in the TPC-A benchmark are the throughput (tpsA) and price-per-tps. The throughput indicates the number of transactions per second with the TPC-A benchmark; the price/performance is the total system cost divided by the throughput. This second metric is designed to demonstrate the value of the system.

The TPC-A benchmark was one of the first standardized benchmarks to provide real and fair comparability between systems. The TPC-A workload is quite intensive and is still useful as a workload generator for system vendors. Because the computer industry evolves quickly, the TPC-A benchmark no longer represents today's workloads, and as such has become obsolete.

TPC-B

The TPC-B benchmark was adopted in 1989 as a follow up to the TPC-A benchmark. Based on the TP1 benchmark or Credit-Debit benchmark, the TPC-B measures performance in an update-intensive environment as is typical of OLTP applications.

The TPC-B benchmark also simulates a banking system. Information is stored for each account, teller, and branch in this system. During a transaction, an account is either credited or debited. The corresponding teller and branch must be updated to reflect the transaction. This benchmark is characterized by the following elements:

- ◆ Significant disk input/output
- ◆ Moderate system and application execution time
- ◆ Transaction integrity (ACID properties)

The TPC-B benchmark employs only one update-intensive transaction type to load the system. Although this transaction type does reflect some of the workload generated by OLTP transactions, it does not reflect the entire range of OLTP environments. Because of the lack of emulated users, the TPC-B workload is seen as more of a stress test than an actual OLTP simulation. The TPC-B benchmark became obsolete on June 6, 1995. From that date on, no more TPC-B results can be published. As of December 6, 1995, all TPC-B results were removed from the official TPC results list.

Even though this benchmark has officially been retired, it is still used internally within the computer industry. The TPC-B benchmark is an excellent system stress test and is easily configured and run.

Although the TPC-B benchmark shares the same transaction profile and database schema with the TPC-A benchmark, the two cannot be compared. The TPC-B benchmark is a batch mode benchmark that does not simulate users as does the TPC-A benchmark.

The TPC-B benchmark is scaled based on the performance reported; the larger the result to be published, the larger the size of the database. A 50 tpsB database has a size of 0.5GB; a 500 tpsB database has a size of 5GB (just as with the TPC-A benchmark). This configuration must also include enough disk space to provide for 30 days of online disk storage at the rate measured. This requirement increases the size of the system being priced.

The metrics used in the TPC-B benchmark are the throughput (tpsB) and price-per-tps. The throughput indicates the number of transactions per second with the TPC-B benchmark; the price/performance is the total system cost divided by the throughput. This second metric is designed to demonstrate the value of the system.

The TPC-B workload is intensive and is still useful as a workload generator for system vendors. The lack of front-end terminals makes the TPC-B benchmark a much easier workload generator to set up and run. Because the TPC-B benchmark is seen as not representing today's workloads, it has become obsolete.

TPC-C

The TPC-C benchmark, adopted in July 1992, is the mainstream benchmark of the TPC today. The TPC-C benchmark simulates an OLTP workload as did the TPC-A—but the transactions are more complex and the benchmark actually has multiple transaction types. The TPC-C is similar to the TPC-A benchmark in that it is a full-system emulation simulating a multiuser environment.

The TPC-C benchmark models an order-entry system made up of a number of warehouses, each with associated districts that take orders. Orders are taken and delivered, payments are made, and account queries occur. The TPC-C benchmark is the first TPC benchmark to require input/output screen formatting. The TPC-C benchmark is characterized by the following elements:

- ◆ Multiple online terminal sessions
- ◆ Significant disk input/output
- ◆ Moderate system and application execution time
- ◆ Transaction integrity (ACID properties)
- ◆ Nonuniform distribution of data access through primary and secondary keys
- ◆ A database consisting of many tables with a wide variety of sizes, attributes, and relationships
- ◆ Contention on data access and update

The TPC-C benchmark employs five different transaction types that stress different areas of the system. Each of these transactions have different criteria to which they must adhere. The five transactions are listed here:

- ◆ **New-Order.** The New-Order transaction enters a complete order of 5 to 10 line items through a single database transaction. It is a midweight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy online users. The response time specification requires that 90 percent of the New-Order transactions must complete in less than 5 seconds.

- ◆ **Payment.** The Payment transaction must make up at least 43 percent of the completed transactions in the measurement interval. The Payment transaction updates the customer's balance and updates the district and warehouse sales statistics to reflect the change. It is a lightweight, read-only transaction with a high frequency of execution and stringent response time requirements to satisfy online users. The response time specification requires that 90 percent of the Payment transactions must complete in less than 5 seconds.
- ◆ **Order-Status.** The Order-Status transaction must make up at least 4 percent of the transactions in the measurement interval. The Order-Status transaction queries the status of the customer's last order. It is a midweight, read-only transaction with a low frequency of execution and a response time requirement to satisfy online users. The response time specification requires that 90 percent of the Order-Status transactions must complete in less than 5 seconds.
- ◆ **Delivery.** The Delivery transaction must make up at least 4 percent of the transactions in the measurement interval. The Delivery transaction consists of processing a batch of 10 new, not-yet-delivered orders. Because it is intended that the Delivery transaction be executed in a deferred mode through a queuing mechanism and that the results be written to a log file, the response time requirements are relaxed. The response time specification requires that 90 percent of the deferred Delivery transactions must complete in less than 80 seconds.
- ◆ **Stock-Level.** The Stock-Level transaction must make up at least 4 percent of the transactions in the measurement interval. The Stock-Level transaction determines the number of recently sold items that have a stock level less than a specified threshold. It is a heavy read-only transaction with a relaxed response time requirement. The response time specification requires that 90 percent of the Stock-Level transactions must complete in less than 20 seconds.

These five transactions make up the workload of the TPC-C benchmark. Implementing and running a TPC-C benchmark is time consuming and expensive.

As is true for the TPC-A benchmark, the front-end processing of the TPC-C benchmark requires additional machines to be used to offload the work of the data input/output screen handling. Typically, a Transaction Monitor (TM) is used to multiplex these connections.

The first metric used in the TPC-C benchmark is the Maximum Qualified Throughput (MQTh), which is the number of New-Order transactions per minute; the metric is reported as tpmC. The price-per-tpmC (price/performance) metric indicates the total system cost divided by the MQTh. This second metric is designed to demonstrate the value of the system.

The TPC-C database is scaled based on the performance reported. For each warehouse configured, there are 10 terminals. Each terminal drives the system at a specific rate. Because the input data generation is strictly regulated, it is necessary to increase the number of warehouses configured in the system to have the proper number of terminals driving the system sufficiently.

Typically, it is expected that the performance reported is approximately 11 tpmC per warehouse configured. The priced configuration must also include enough space to store 180 days of growth of the database. This requirement adds significantly to the cost and size of the configuration.

The TPC-C benchmark can be seen as a good test of a system's OLTP performance. The complexity and variance of the workload create an intense workload that can stress any system to its limit.

TPC-D

The TPC-D benchmark, adopted in April 1995, is the first decision support benchmark adopted by the TPC. The TPC-D benchmark simulates a wide array of decision support environments. It is designed to model a global enterprise business environment. The TPC-D benchmark simulates a live production decision support environment using 19 distinct, unrelated queries from a common database. These queries consist of 17 read-only queries and 2 update queries.

The TPC-D benchmark employs a variety of different queries chosen to have broad industry relevance. These queries make heavy use of joins, aggregates, groupings, and sorts and are executed on a range of volumes of data. The TPC-D benchmark is different from previous benchmarks in that it has three metrics associated with the results. The TPC-D queries can be measured in single-stream and multiple-stream modes, each having their own metric as well as the traditional price/performance metric. The TPC-D benchmark is characterized by the following elements:

- ◆ Queries against large volumes of data
- ◆ Queries that exhibit a variety of access patterns
- ◆ Queries of an ad-hoc nature
- ◆ Highly complex queries—far more complex than OLTP queries
- ◆ Generation of intense activity on the part of the server component under test
- ◆ Representative of critical business questions
- ◆ Compliance with specific population and scaling requirements
- ◆ Implemented with constraints derived from staying synchronized with an online production database
- ◆ Transaction integrity (ACID properties)

The TPC-D benchmark consists of 19 distinct, unrelated, complex queries consisting of 17 read-only queries and 2 update queries. The queries are designed to model some of these business tasks:

- ◆ Pricing and promotions
- ◆ Supply and demand management

- ◆ Profit and revenue management
- ◆ Customer satisfaction study
- ◆ Market share study
- ◆ Shipping management

The performance metrics used in the TPC-D benchmark measure different aspects of the capability of the system. These include the size of the database against which the queries were executed; the TPC-D query processing power, QppD@Size (queries run sequentially); and the TPC-D throughput, QthD@Size (queries run concurrently). The QppD metric represents the Query Processing Power for the TPC-D benchmark result; the QthD metric represents the Query Throughput for the TPC-D benchmark result. The price/performance metric is defined as QphD@Size and is based on a composite query-per-hour rating derived from both QppD and QthD. The QphD metric represents the Price per Query per Hour for the TPC-D benchmark result. To be compliant with TPC policy, all three metrics (QppD@Size, QthD@Size, and QphD@Size) must always be expressed as a set. Furthermore, the TPC believes that comparing a TPC-D result run on a database of a certain size is not comparable with a TPC-D result run on a database of another size.

The TPC-D database is not scaled according to the performance you achieve (as is true with the other TPC benchmarks); rather, the benchmark sponsor chooses the size of benchmark with which it wants to work. This benchmark size must be reported as part of the benchmark metrics. The allowed sizes are 1GB, 10GB, 30GB, 100GB, 300GB, and 1000GB.

The TPC-D is important because it has defined a complex workload that models a decision support system. The TCP-D benchmark allows you to judge the performance of systems on something other than OLTP results alone.

TPC-E

The TPC-E benchmark is designed to quantify the ability of a system to support the computing environment appropriate to large business “enterprises.” These environments typically support workload demands that exceed the demands imposed by other TPC benchmarks.

NOTE: At the time this book goes to press, the TPC-E benchmark is still under development; it has not yet been accepted as an official benchmark (although I believe it will be accepted soon).

The size and complexity of the TPC-E benchmark database operations far exceed those of any other TPC benchmark. The TPC-E benchmark is characterized by the following elements:

- ◆ Queries against large volumes of data
- ◆ Support for a large user community with strict response times

- ◆ Support of concurrent batch execution while retaining online user response times
- ◆ Support of a large and complex database image accessed with both high frequency and high throughput operations
- ◆ Stress on sort processing
- ◆ Demonstration of the ability to perform large-scale database updates
- ◆ Demand for proof of concurrent backup abilities
- ◆ Recovery from system failure
- ◆ Transaction integrity (ACID properties)

As with the TPC-A and TPC-C benchmarks, the TPC-E benchmark actually simulates user access through terminal devices. This arrangement provides a multiuser, full-system emulation. The TPC-E benchmark requires input/output screen formatting as does the TPC-C benchmark.

As is the case with the TPC-A and TPC-C benchmarks, the front-end processing of the TPC-E benchmark requires additional machines to be used to offload the work of the data input/output screen handling. Typically, a Transaction Monitor (TM) is used to multiplex these connections.

The TPC-E benchmark consists of a combination of both OLTP and batch components, made up of the five TPC-C transactions, two additional OLTP transactions, and a concurrent batch activity. The new OLTP transactions (in addition to the five TPC-C transactions) are listed here:

- ◆ **Customer-Inquiry.** This transaction type represents a real-time customer service query that returns account information. It is frequently accessed and has stringent response time criteria.
- ◆ **Customer-Status.** This transaction type represents a real-time corporate-wide Management Information Systems (MIS) query that extracts customer payment information. This transaction is a heavy read-only transaction executed at a low frequency and has relaxed response time criteria.

The third new transaction is the **Customer-Demographics** transaction, which represents a real-time corporate-wide MIS query that evaluates customer buying activity based on demographic factors. This query is characterized by a heavy read-only transaction that is infrequently executed but consumes significant resources. The response time requirement is relaxed for this query.

The first metric used in the TPC-E benchmark is the Maximum Qualified Throughput (MQTh)—the number of New-Order transactions per minute, which is reported as tpmE. The and price-per-tps (price/performance) metric indicates the total system cost divided by the MQTh. This second metric is designed to demonstrate the value of the system. Along with these primary metrics, there are several secondary metrics such as CPU busy, system I/O rates, and so on.

As with the other OLTP benchmarks, the database size scales with throughput. This requirement provides for a fair comparison between systems by requiring larger systems to access larger databases.

The TPC-E benchmark will probably see less activity than the other TPC benchmarks because of the tremendous cost and effort required to execute it. In addition, I think that some companies will use the TPC-E benchmark while others migrate to the TPC-C/S or TPC-Server benchmark, depending on their market focus.

TPC-C/S

The TPC-C/S (Client/Server) benchmark is an OLTP benchmark designed to reflect the performance and price characteristics of the client/server solution to OLTP problems.

NOTE: At the time this book goes to press, the TPC-C/S benchmark is still under development; it has not yet been accepted as an official benchmark (although I believe it will be accepted soon).

The TPC-C/S benchmark was developed to test system configurations designed with a client/server solution in mind. A *client/server solution* refers to the division of work between the client (front-end) and server (back-end) machines. The TPC-C/S benchmark is characterized by the following elements:

- ◆ Use of modern Graphical User Interfaces (GUIs) to improve user friendliness and end-user productivity
- ◆ Use of application development tools to speed development time and promote ease of maintenance
- ◆ Use of decision support and office automation tools
- ◆ Significant network activity
- ◆ Access to data when the server/host is not available
- ◆ A move closer to the new, decentralized business model
- ◆ Provisions for the user to access the large number of applications available for workstations and personal computers
- ◆ Exploitation of the perceived price/performance advantage of the client/server solution
- ◆ Use of online and deferred execution modes
- ◆ Use of multiple online sessions
- ◆ Significant disk input/output
- ◆ Significant client processing of multiple data types: numeric, text, graphics, and images

- ◆ Transaction integrity (ACID properties)
- ◆ Parallel processing by the client and server

As with the TPC-A, TPC-C, and TPC-E benchmarks, the TPC-C/S benchmark actually simulates user access. Unlike the other benchmarks, the TPC-C/S benchmark simulates that access through a workstation GUI. This arrangement provides a multiuser, full-system emulation. The TPC-C/S benchmark requires input/output screen formatting through this interface.

The performance metric reported by the TPC-C/S benchmark is a “business throughput” measured by the number of New-Order transactions per minute. The metric is the tpmCS (transaction per minute C/S). As with all the other TPC benchmarks, the associated price/performance must also be disclosed.

With the rapid emergence of client/server computing, the TPC-C/S benchmark may soon overtake the other TPC benchmarks in popularity. The TPC-C/S benchmark—as with most other TPC benchmarks—is very time consuming and expensive to run.

Results of TPC Benchmarks

The current set of TPC results is available to the public from the TPC Web site:

<http://www.tpc.org/>

The Web site has information on the TPC benchmarks themselves as well as individual and complete results. Individual results are in the form of the Executive Summary. The Executive Summary is usually found in the front section of the Full Disclosure Report and contains the performance metrics, a listing of the hardware and software used in the benchmark, and a price breakdown. The Executive Summary also includes a diagram of the benchmarked configuration.

The TPC results spreadsheet is available in SYLK, Adobe PDF, and Excel formats and contains a complete list of the current TPC benchmark results. As benchmarks become obsolete (as was the case for the TPC-A and TPC-B benchmarks), the results are removed from the official list. As new versions of benchmarks become available, there is usually a lag time during which the results from the old versions remain on the spreadsheet (eventually, the old results fall off the list). In some cases—such as for the change from TPC-C version 2.x to 3.0—benchmark results can be upgraded for a limited time.

Interpreting the Spreadsheet

I have refrained from inserting a copy of the spreadsheet in this book because it would have been out of date before the book left the printer. However, I do want to explain how to interpret the spreadsheet. Although the format of the spreadsheet changes infrequently, the concepts remain the same.

The spreadsheet is divided into sections that separate the results for each benchmark and major revision. It usually starts with new results, followed by the individual result sets in alphabetical order. Within each section, the results are sorted alphabetically by sponsor. The following information refers to the System Under Test (SUT) used in the benchmark. To get information about front-end machines, you must look at the Executive Summary for that benchmark. The following list describes the columns in the results summary spreadsheet:

- ◆ *Company*: The sponsor of the benchmark. Typically, a benchmark has more than one sponsor (perhaps a hardware company and an OS company, of which one is considered the primary sponsor).
- ◆ *System*: The name and model number of the benchmarked hardware.
- ◆ *Spec Revision*: The version of the specification under which the benchmark was published. Because minor revisions are comparable, they are listed together.
- ◆ *Throughput*: The primary performance metric. The performance of the system.
- ◆ *Price performance*: Another primary metric.
- ◆ *Total system cost*: The priced components are determined in each benchmark specification.
- ◆ *Database software*: The name and version of the database software used in the benchmark.
- ◆ *Operating system*: The name and version of the operating system used in the benchmark.
- ◆ *TP monitor*: The name and version of the TP monitor used in the benchmark.
- ◆ *Company*: The sponsor of the benchmark. This column is duplicated because the width of the spreadsheet usually forces the table to be broken into two sections for printing.
- ◆ *System*: The name and model number of the benchmarked hardware. Duplicated for purposes of printing the spreadsheet.
- ◆ *Cluster*: Indicates whether the benchmark configuration is configured as a cluster or a single machine.
- ◆ *MP/UNI*: Indicates whether the server is a uniprocessor or multiprocessor computer.
- ◆ *Qty/Processors/MHz*: Describes the number, type, and speed of the CPUs in the system.
- ◆ *Original received*: The date on which the FDR was received at the TPC. At this time, the results can be announced by the sponsor.
- ◆ *Submitted for review*: The date on which 80 additional copies of the FDR were received by the TPC for distribution to the membership.

- ◆ *Date accepted:* The date on which the 60-day challenge period ended. If a challenge is pending, this date is held up.
- ◆ *Prices updated:* Indicates that a pricing change has occurred to the FDR.
- ◆ *Qrt Rpt issue:* Indicates the quarterly report of the TPC in which the benchmark results first appeared.

Having served as a representative on the TPC for my employer, I can attest to the hard work and dedication offered by each of the participants. The results of the organization itself stand as a tribute to the people who have worked so hard to make it a success. One thing that has always impressed me is that even though the benchmark subcommittees are made up of people from competing organizations, there is a great deal of cooperation and friendship among the members.

Although industry standard benchmarks cannot predict how your application will perform on a particular platform, the benchmarks are a good indicator of how competing platforms compare in these environments. Hopefully, TPC results can help you narrow down the choices to make your purchasing decision easier. Of course, the best indication of how well a particular application will run on a particular system is to benchmark it yourself. The process of developing a custom benchmark is briefly discussed later in this chapter.

Publication Benchmarks

Several publications have designed and built their own tests for benchmarking RDBMS products. Typically, the publication provides a good description of the test in the article that uses the results (*Ziff-Davis* labs and *PC Week* labs both do these types of tests). Many of these tests are very good and can provide you with insight about the performance of specific areas of RDBMS performance such as loading, backups, and so on.

One thing to look for in these independent tests is the criteria that has been set up for the hardware and RDBMS products. Unlike the TPC benchmarks, these tests have a somewhat different goal in mind. Typically, publication benchmarks target a specific comparison (such as several different RDBMS products running on the same hardware or several hardware platforms running the same OS and database).

These tests can be useful to compare systems but may not always show the most optimal configuration for that platform. Look at these tests carefully and decide whether they apply to you and give you useful information. Decide whether the tested configuration is realistic for you. If so, the results of these tests can be very useful.

Custom Benchmarks

Custom benchmarks are designed to create a workload that reflects the workload your user community will be generating. It is often difficult to predict the workload of hundreds or perhaps thousands of users, each with their own agenda. Although most custom benchmarks are designed by the end-user for a particular installation, there are some prepackaged custom benchmarks designed to generate a workload for a specific application.

If you are purchasing a packaged application, see whether there is a workload generator already built for that product. If there is, you can save yourself months of work designing and implementing such a workload generator. If one is provided, make sure that there is enough variance in it to allow customization for your installation.

Your custom benchmark should be designed to allow you to simulate the load of the number of users you support in production. It should also provide timing information that is critical for response time and throughput analysis. If your installation is required to support 1,000 concurrent users with 2-second response time criteria, it is important that the benchmark be able to simulate this.

When you put your system into production, it is too late to determine that you cannot support the required number of users, or that a checkpoint brings the system to its knees. Test these elements before the system rolls out into production.

If you are implementing a client/server system, make sure that you simulate the load over a network so that you can test the entire system. If possible, your benchmark should be an end-to-end test, simulating not just the workload but the display processing as well. Several vendors on the market today sell keystroke generators for character-mode applications as well as client/server load generators. Products such as Empower from Performix can be used to simulate thousands of users relatively easily.

Writing Your Own Benchmark

Writing your own benchmark is by far the best way to analyze the performance of a particular platform as well as to assist in the development and tuning phases of your application implementation. As in any project, writing a benchmark consists of the design, implementation, and analysis phases—each of which has its own special concerns. Of particular importance is to make sure that the proper workload is generated.

A custom benchmark can consist of an end-to-end test in which user keystrokes are emulated and data output examined or it can consist only of generating an equivalent workload on the server. Although the end-to-end test much more accurately represents the final workload, such a test is often too difficult to engineer. It is especially difficult to emulate a large number of PCs in a client/server configuration—but products are emerging on the market designed to help with this problem (consider Empower C/S from Performix).

These benchmarking/testing products are designed to capture a single transaction and let you programmatically simulate hundreds and perhaps thousands of similar connections from a set of driving machines. These products allow you to randomize inputs and manage individual user contexts while simulating the load.

If you are just implementing a server workload test, be sure to carefully test the capacity of network connections and user contexts to ensure that your product roll out is not a failure. A robust server that can handle an intense workload does not guarantee that the configuration can handle hundreds or thousands of user connections. These factors are important to keep in mind during the design stage.

Design

The design stage is perhaps the most important phase in the development of a custom benchmark. Creating a workload that does not represent the load generated by the user community does not result in a successful implementation. In designing a custom benchmark, consider the following factors:

- ◆ The workload should match that of the user community that will be using the final implementation.
- ◆ Specify response time criteria.
- ◆ Build in other essential functions such as backups and other administrative tasks.
- ◆ If possible, generate the same network load as the final system.
- ◆ Take into account peak loads such as shift changes.
- ◆ Add a sufficient amount of randomness to vary the workload.
- ◆ If possible, use the actual application that will be used in the final configuration.
- ◆ Account for growth in the number of users and the work generated by them.

These are a few ideas you should consider in your custom benchmark. After completion of the design phase, you will want to implement the benchmark in the most efficient way possible.

Implementation

During the implementation phase, the benchmark is developed and run on the benchmark configuration. During this phase, it is important to take careful notes on the progress of the benchmark. Any changes made in the design or implementation should be noted.

It is here when the benchmark is actually run. The performance data is retrieved and analyzed. Be sure to collect as much data as you can to ensure that the analysis is complete. Look for data on such things as these:

- ◆ CPU utilization: Be especially aware of idle CPU cycles.
- ◆ Disk I/O: Look for hot spots in certain tables and disks.

- ◆ Network bandwidth: Don't exceed the network limitations.
- ◆ Response times of transactions: It is important to make sure that you can meet the required response times.
- ◆ Transaction throughput: How much work can really get done?
- ◆ Is the benchmark operating as specified? Check to make sure that you are really driving the system the way you designed the benchmark to do.

Analysis

During the analysis phase, you should look at two things: First, is the custom benchmark performing as specified? Does it generate the workload you want or is there something wrong? Second, how does the system perform?

If the benchmark contains a design flaw or a coding bug, it is essential to determine this as soon as possible rather than proceeding with decisions based on bad data. Verify that the results look like what you expect them to; if they don't, ask the question *why*?

Once you verify that the test is working properly, it is time to analyze the results. If you are running the test to determine which competing product to buy or to determine whether an application change has an effect on the system, it is important to retune the system after any changes are made. Running the same test on two different hardware platforms may give deceiving results if you don't analyze the data correctly. Here's an example:

By using the same tuning parameters on System A and System B, you may see better results on System A because of an architectural feature in the hardware. By tuning both systems individually to their highest potential, you may find that System B performs much better. Each system is different; it is invalid to hold tuning parameters constant for comparisons. The only way to effectively compare systems is to tune each as well as you can.

Summary

This chapter looked at the industry standard benchmarks from the TPC, briefly described the publication benchmarks, and outlined how to design and implement custom benchmarks. Both standard and custom benchmarks serve a dual purpose: an indicator of performance and a verification tool when changes have been made. You should now have an idea about the effort required to produce a TPC benchmark as well as the effort necessary to implement your own benchmark. These tools can be very powerful and useful once they have been implemented.

This chapter also described the various TPC benchmarks available and how to interpret those results. The TPC benchmark results are available by accessing the TPC Web site and can be a powerful tool in helping you decide which platform is right for you. Because TPC benchmarks are published by the vendors who have a lot at stake, you can be assured that the configurations are optimized and well tuned. Similarly, for custom benchmarks, you should ensure that each system you are comparing is equally well tuned.

Chapter

Performance Monitoring Tools

Before you configure a system for optimal performance, it is essential that you properly analyze the system. Without having good data about the performance and internal workings of the system, it is difficult to know what to modify. This is true for the hardware, the operating system, and the RDBMS.

This chapter gives you a brief introduction to some of the performance monitoring tools available on the market today. These tools may or may not be useful to you; this short overview will help you make that decision.

Performance monitoring tools fall into several categories: Some tools perform an analysis; other tools display system data; yet others are designed to trap certain error conditions. Each has a different focus and each is valuable to the system administrator. In addition, many good application development tools are available today; Chapter 35, “Using GUI Builders,” details some of these tools, including Oracle Developer/2000, PowerBuilder, Delphi, and others.

Many tools are available to help you design, administer, and monitor your system. You can pick up any database magazine and read about many excellent products. Rather than go into a long discussion on these products, this chapter points out what to look for in some of these tools. It also looks at what is already available in your system to monitor performance and what you can obtain from Oracle.

Oracle stores a wealth of information about how the system is performing in the dynamic performance tables. Many tables make up the dynamic performance tables (refer to your Oracle administrator's guide). Never access these tables directly; always access them through the views defined in CATALOG.SQL. Table 6.1 describes a few of the views that are of significance to performance tuning.

Table 6.1 Dynamic Performance Tables Valuable in Performance Tuning

<i>View</i>	<i>Description</i>
V\$ACCESS	Information about locked objects in the database and who is accessing them.
V\$CIRCUIT	Information about virtual circuits. A <i>virtual circuit</i> is a connection into the database through dispatchers and servers.
V\$DB_OBJECT_CACHE	Database objects (tables, indexes, clusters, synonym definitions, PL/SQL procedures and packages, and triggers) cached in the library cache.
V\$DISPATCHER	Dispatcher process information.
V\$FILESTAT	Information on file read/write statistics. This information can be enhanced by setting the parameter <code>TIMED_STATISTICS</code> equal to TRUE. Because setting this parameter degrades performance, use it only if you feel it is necessary to debug a problem.
V\$LATCH	Information about each type of latch; used to indicate latch contention.
V\$LIBRARYCACHE	Statistics about library cache management. Look here for information about the number of library cache hits.
V\$LOCK	Information about locks and resources; does not include information about DDL locks.
V\$QUEUE	Information about the multithreaded message queues. Look here for information necessary to determine average wait time per item.

<i>View</i>	<i>Description</i>
V\$REQDIST	Histogram of request time, divided into 12 buckets; can provide valuable information about request times.
V\$ROLLSTAT	Statistics for all online rollback segments. Look here for information about whether rollback segments are properly configured.
V\$ROWCACHE	Statistics for all data dictionary activity.
V\$SESSION_WAIT	List of resources or events for which active sessions are waiting; can be useful in debugging a particular problem.
V\$SESSTAT	The current statistic values for each current session; used with V\$STATNAME.
V\$SESS_IO	I/O statistics for each user session.
V\$SYSSTAT	The current system-wide value for each statistic in V\$SESSTAT; also used with V\$STATNAME.
V\$WAITSTAT	Block contention statistics. This information is stored only when the parameter <code>TIMED_STATISTICS</code> is set to TRUE. Setting this parameter causes extra overhead and degrades performance; use it only if you feel it is necessary to debug a problem.

The dynamic performance tables include many more tables than are mentioned here. Most tables are used internally and are not of use in performance tuning unless you are debugging a specific problem.

Oracle Tools

Oracle offers several performance monitoring tools. You automatically get SQL*DBA as part of the Oracle RDBMS product, which includes a monitor interface into the V\$ tables. You can also get Server Manager from Oracle (which eventually will replace SQL*DBA as the standard administrative tool). Server Manager (introduced in Oracle version 7.1) is similar to SQL*DBA but provides an optional Graphical User Interface version.

Oracle recently introduced SNMP (Simple Network Management Protocol) agents. These agents (as well as any of the third-party vendors who have been working with Oracle) provide an interface into the V\$ views by using an SNMP management console. Also new from Oracle is a graphical version of SQL Trace. The SQL Trace tool is designed to help track down and fix inefficient SQL statements in your application. Of course, the EXPLAIN PLAN command is always useful in analyzing SQL code.

SQL*DBA Monitor

You can use the SQL*DBA monitor command to display real-time information about locks, enqueue, processes, file I/O, and other system statistics. Monitor is used as a shortcut to display information stored in the V\$ tables. Monitor has been around as long as SQL*DBA; with the emergence of Server Manager, Monitor may not be of use much longer.

Server Manager

Server Manager is now available on several platforms. Server Manager is similar to SQL*DBA in that it is designed to assist in the administration of the Oracle database. But where SQL*DBA is character based, Server Manager provides a Graphical User Interface. From Server Manager, you can display much of the information available in the V\$ tables.

Oracle SNMP Agents

Oracle SNMP Agents are available in Oracle version 7.3 and later. The Oracle SNMP agents allow third-party vendors to easily retrieve performance data from Oracle. These SNMP agents in conjunction with OS SNMP agents can provide a complete management solution for your system. Oracle has teamed up with other members of the System Management Tools Initiative (SMTI) to provide a shrink-wrapped solution to your RDBMS management needs.

When you use a performance monitoring tool that uses the Oracle SNMP agents, you can avoid selecting information from the V\$ tables directly. When you access the V\$ tables, you cause some overhead within the database itself. When you use the Oracle SNMP agents, this information is retrieved in a more efficient manner (because Oracle designed it specifically for the RDBMS).

The Oracle SNMP agents are designed not only to give you reliable and pertinent information, but to cause as little overhead on the system as possible. Most SMTI and other third-party vendors will soon begin collecting their information through the Oracle SNMP agents (if they have not already done so).

SQL Trace

Oracle's SQL Trace facility provides performance information about individual SQL statements. SQL Trace is activated by executing the following command:

```
SQL> ALTER SESSION SET SQL_TRACE = TRUE;
```

SQL Trace is turned off with this command:

```
SQL> ALTER SESSION SET SQL_TRACE = FALSE;
```

During the time that SQL Trace is active, performance statistics for all SQL statements generated within this session are stored in a trace file. You can then use the Oracle utility TKPROF to convert the trace file into readable form. TKPROF can also be invoked with the EXPLAIN PLAN option if you want to additionally display the execution plan. The following statistics are gathered for each SQL statement:

- ◆ Parse, execute and fetch counts.
- ◆ CPU and elapsed times.
- ◆ Physical reads and logical writes. Remember that the DBWR process does the writes in a deferred manner.
- ◆ Number of rows processed.
- ◆ Library cache misses.

SQL Trace can be enabled for all users by setting the initialization parameter SQL_TRACE to TRUE. The information gathered by SQL Trace can be quite useful in debugging inefficient SQL statements, as you will see in detail in Chapter 25, “Using EXPLAIN PLAN and SQL Trace.”

EXPLAIN PLAN

The Oracle EXPLAIN PLAN command displays the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. By examining the execution plan, you can see exactly how Oracle executes your SQL statement. The execution plan can help you determine whether you have written the an efficient SQL statement or whether changes can be made to optimize the statement.

To execute an EXPLAIN PLAN statement, you must first create a table with the name `plan_table` and with the specified `plan_table` format. The table format and the SQL statement required to create the table are included in Oracle in the SQL script UTXPLAN.SQL. Once this table is created, you can execute the EXPLAIN PLAN statement by issuing these SQL statements followed by your SQL statements:

```
EXPLAIN PLAN
SET STATEMENT_ID = 'NAME'
FOR
```

In this syntax, NAME specifies a label for the statement in the `plan_table` table. Whether you execute it on its own or in conjunction with the SQL Trace facility, the EXPLAIN PLAN command provides useful information about how SQL statements can be optimized.

OS Tools

Every operating system has a set of tools to help you monitor the system. Although the tools themselves differ depending on whether you are running UNIX, NetWare, NetWare NT, OS/2, or some other OS, each tool gives the same basic information to the administrator. The values that the OS monitors should include the following basic parameters:

- ◆ **CPU utilization.** In an Symmetric Multiprocessor (SMP) environment, this information should be broken down by CPU.
- ◆ **Disk I/O.** This information should be reported on a per-disk basis (or on a per-volume basis if you are using a disk array).
- ◆ **Memory utilization.** This statistic should indicate the amount of physical memory used and available.
- ◆ **Paging statistics.** If you are using a virtual memory operating system, this statistic should be available.
- ◆ **Context or process switches.** For any multitasking OS, this switch information is important.

These are but a few of the important indicators you will require in the upcoming chapters. Although each operating system is different and has different monitoring tools, the basic concepts are the same.

Third-Party Tools

Many third-party tools on the market today are excellent. You should take several factors into consideration when shopping for a third-party monitoring tool: Does it fit into your environment? Does it put an undue burden on the system? Does it monitor all the parameters you need?

Some products are designed with a particular operating system in mind; some support many different environments; still others monitor only Oracle through SQL*Net and consider the server's operating system irrelevant. Be sure that the monitoring tool you purchase fits into your environment. If possible, get a demonstration version and try it out before you buy.

Make sure that the monitoring tool does not put a heavy load on the server. If configured improperly, some products continually extract data from the Oracle internal performance tables, causing undue contention in the system. Try running a benchmark with and without the performance monitor running. Perhaps the collection interval is too fast and can be reduced. It is not necessary to extract performance information from the system every second when a 30-second interval will do. Reducing the collection interval can significantly reduce the load on the system.

NOTE: Performance monitoring tools that offer a graphical display showing real-time information can sometimes affect the performance of your system. Tools that graphically update performance information on a per-second basis can be especially hard on the system. Remember: If the tool displays an update every second, it is querying the database at that rate.

Here are some questions you should ask yourself when purchasing a performance monitoring tool. Remember that you will have your own individual needs based on your implementation.

- ◆ Does the monitor display all the information you need?
- ◆ Do you need OS and RDBMS monitoring or is RDBMS monitoring sufficient?

Most third-party tools available today are sufficient for most needs. Be sure to shop around to find the one that is right for you. I have intentionally avoided making any particular recommendations here because there are many good tools on the market today—and they are continually improving.

By shopping around, you will find a performance monitoring tool that fits into your existing network and management strategy. Don't judge a tool based on how "pretty" it looks; look at what it monitors and how it gets its information.

The tools available today fall into one of several categories: those that simply monitor parameters and display them on a graphical console, those that alert you when they reach some threshold, and those that are combinations of these first two. The one you need depends on your configuration.

Real-Time Monitors

The real-time monitors usually have a graphical display and provide histograms, or line charts, of various parameters. Real-time monitoring tools are useful for debugging a resource problem or for tuning an application. With these monitors, you can watch the display while you are running to see whether any particular resource is being overused or whether a certain area of the database is experiencing extremely high activity.

I recommend a real-time monitoring tool for debugging applications, trying new tuning parameters, or comparing different applications.

Because of the type of monitoring being done, these performance monitoring tools typically look at Oracle's V\$ tables or maybe even the internal X\$ tables. Because this can have an adverse affect on the system's performance, use these types of tools periodically to check the health of the system or when you are debugging a problem.

Another alternative is to run these tools with a fairly long sample interval (such as 30 seconds or a minute) to reduce the overhead on the system. To monitor the system for a particular problem, consider a threshold monitor. Threshold monitors are used when you don't think you have any problems; they wake up and alert you of some potential or real problem.

Threshold Monitors

Threshold monitoring tools remain quiescent until some sort of condition requires an alert. These conditions are usually an indication that an event (such as a tablespace filling up) has occurred or that some sort of failure needs attention.

Threshold monitors typically work on a sampling interval; for some events, they may insert into the database itself triggers that are fired when some sort of event occurs. You should set the time interval for sampling on a per-event basis. If there is a critical event, you may want to shorten the sampling interval. In general, the longer the sampling interval, the less effect there is on the system's performance.

Many of the threshold monitors available today can monitor a variety of RDBMS products and applications as well as a variety of different operating systems. In a large installation, it can be valuable to monitor all your systems from one console, regardless of what OS and products the system is running.

If your RDBMS system performs a mission-critical task, it may be imperative that you employ a threshold monitoring program. The most sophisticated of these tools have options that give alerts in several forms, including graphical display, e-mail, and even paging. This type of monitor may be able to alert you in advance of a problem, allowing you to correct the problem and avoid downtime.

Summary

This chapter gave you a brief introduction to performance monitoring tools. Whether you use third-party tools or the tools Oracle provides in combination with the tools that come with your operating system, it is important to monitor your system regularly.

When you implement your configuration, it is important to monitor the system during the development and tuning phases. It is equally important to monitor your system on a regular basis. The worst type of problem is one you are not expecting and are not prepared for.

Monitoring tools that monitor both RDBMS and OS parameters and alert you in the event of a problem are very useful. These tools can help you find problems before they occur and can help you avoid costly downtime. If your system is used in a mission-critical application, the time spent recovering from a failure can be devastating; monitoring such systems is imperative.

If you want to use a third-party monitoring tool, choose one that has the functionality you need for your particular installation. Consider getting a monitoring tool that monitors both the OS and Oracle. Finally, make sure that the performance monitor does not adversely affect the performance of your system.

By choosing the right performance monitor, you should be able to assess changes made to your system as well as monitor the day-to-day health of the system. Be sure to monitor your system periodically even if everything seems fine. By checking the health of the system regularly, you may be able to catch small problems before they become big ones.

Chapter—

Performance Engineering Starts at the Design Stage

The most effective way to achieve an optimally performing system is to start at the design stage. In the design stage, you have the most control over what the application will be at the rollout stage. In the design stage, any optimizations that can be made to table layouts, disk configurations, system CPUs, and memory will have the most effect. Changing these components after the system has already been in production can be costly and time consuming.

Of course, not all performance engineering can be done in the design stage. Nobody can perfectly model the users who will access the system or the variances in load over time. These factors call for corrective action after the system is in test and in production, but having engineered performance into the system from the start will make this task easier.

Design Stage

In the design stage, it is important to look not only at the application and the database but at the sizing of the system as well. This chapter looks at some of the ways you can design performance into your system from the very start. Some of the key areas are the layout of the database itself, the use of indexes and clusters, proper design of the application, and sizing of the hardware—including the network.

Database Layout

You can divide the database layout into two sections. The first issue to consider in laying out the tables is the relationship between the data on the system. This relationship determines how the tables are created and which tablespaces they are part of. The second issue to consider in laying out the tables involves their proper placement on the physical disks. The placement of data files on the disks is critical in properly balancing I/Os to avoid a disk bottleneck.

Many tools are available to help you in the design stage. Oracle offers the Designer/2000 (there are many competing tools on the market today) that can help you define the relationships between tables very effectively.

Oracle Designer/2000 is a sophisticated tool that, in conjunction with Oracle's Developer/2000 development tools, can take you from the business model to the production database. Designer/2000 starts with business modeling that assists with the design of the physical database. Designer/2000 provides both system and process modeling; it is also a design tool that can assist you with the layout of the database, stored procedures, triggers, and so on.

The resulting data generated by the Designer/2000 tools can be used by Developer/2000 to generate applications. Design tools such as Designer/2000—in conjunction with Developer/2000—can improve productivity and enhance the design and performance of the resulting system. Most production databases are extremely complex with many inter-table links. These tools can take some of the complexity out of the design stage and make the database layout more efficient.

Part II of this book, “Tuning the Server,” details why it is important to properly lay out the data to separate random and sequential I/Os. That part of the book also describes how disk arrays can help with this task. It also looks into separating data files from their associated indexes.

Part III of this book, “Configuring the System,” details how different applications call for different configurations. It looks at different workloads such as OLTP, decision support, and batch processing. That part of the book looks at how each design is different and how each can take advantage of the workload characteristics of the system.

Indexes and Clusters

The design phase is the time to decide how to make the most effective use of indexes and clusters. Because the effectiveness of indexes and clusters is related to both the physical design of the database and the application, the design stage is when you can affect both of these elements. Sometimes, it is necessary to re-analyze the database and application to determine whether indexes and clusters are being used correctly; this is not necessary if they are properly designed.

Indexes and clusters are effective only when the application accesses the database in a manner that allows for the use of these features. By properly designing indexes and clusters based on the data access that is required, you can optimize the use of indexes and clusters. To properly optimize your system, design both the application and database based on your knowledge of how this data is to be accessed.

Chapter 10, “Performance Enhancements,” looks at the most effective use of indexes and clusters and how to determine whether you can benefit from using them.

Application Design

The design of the application can also be a deciding factor in the future performance of the system. Carefully plan the application to ensure that deadlocks are avoided and that there is no contention on some limited resources. For example, if your application has everyone accessing the same tables, you may cause undue contention. If your front-end system has excess processing power, take advantage of it by performing input validation there rather than relying on the RDBMS to do it for you.

Look for where the resources will be most in contention and try to design the application to offload work from these areas. Analyze the SQL calls to check for excessive table scans where they may not be necessary. Try to reduce the amount of data sent over the network by taking advantage of stored procedures and packages. Part IV, “Tuning SQL,” looks at these things in detail.

Hardware Sizing

When you decide what hardware to use for your application, make sure that there is sufficient power to handle the load. Often, the workload is underestimated or increases at an unexpected rate. Be sure that you have an upgrade path. Here are some factors to keep in mind when determining the hardware you will use:

- ◆ Can the CPUs be upgraded? Can you put in more or faster CPUs?
- ◆ Can the I/O system be upgraded? How easy is it to expand? Can you add more disks and controllers easily?

- ◆ How much memory does the system support? (If you are already at the limit at rollout, there is no room for expansion.)
- ◆ Can you add more network controllers? (Often, you will want to add a second network for backups or to increase user bandwidth.)

Don't buy more hardware than you need, but if you need to upgrade, make sure that you will not have to start from scratch. Upgrading components is always much less expensive than investing in a new system.

Network Considerations

Make sure that your application will not exceed the bandwidth of the network. During testing, the load on the network is often much less than the load generated by real users. It is difficult (not to mention expensive and time consuming) to segment a network after the application has already been in production.

Much testing is done on applications by loading the server with a small number of clients with a heavy workload. Although this arrangement can offer a lot of good data for tuning, remember that the system is stressed in a different way if you don't have a large number of users. In practice, you may find that you have to increase the number of dispatchers or tune some OS parameters to handle those additional connections.

Performance Tuning after the System Is Built

After the system is built, the job of tuning begins. In the design stage, you engineered performance into the system and you may have a good idea where the bottlenecks or “hot spots” might be. It is only after the system has been designed and built that you can really test your theories.

If possible, generate user input in a laboratory environment and test the system for both stability and performance. It is difficult to generate large numbers of connections—especially in a client/server environment—but if you can generate those connections, the results are worth it. In the testing stage, be sure to keep careful notes about any tuning you do and how those changes affect the system. Remember the five-step method discussed in Chapter 4, “Tuning Methodology”:

1. Analyze the problem.
2. Develop a solution.
3. Set realistic goals.
4. Test the solution.
5. Analyze the results.

By following and sticking to an organized methodology, your results will be worthwhile. Remember that careful note-taking is a must.

Tuning the Client

In a client/server configuration, it is important to make sure that the client is not causing major delays. When using automated GUI application builders, you can sometimes forget that the application can be inefficient. Remember that not everyone in the corporation has the fastest PC on their desk, so optimize the application, if possible.

The nice thing about tuning a client machine is that once it is tuned, it doesn't make any difference how many other machines are on the network. The client is still tuned. Even if you tune the server well, adding more connections can change the workload, making it necessary for you to tune the server again.

Tuning the Server

The server is the most complex aspect of system tuning. Consider that the server may serve hundreds or thousands of connections, that other activities may be occurring on the server, that you have to balance throughput and response times, and so on. Here are some of the factors to keep in mind when tuning the server:

- ◆ Is the database tuned effectively?
- ◆ Is the database layout optimal?
- ◆ Is the OS tuned effectively?
- ◆ Is the hardware sufficient?
- ◆ What other processes are running on the system?
- ◆ What is the balance between users and batch, users and backup?

Keep in mind that the server is a shared resource that must be tuned and balanced to serve the needs of all. Achieving this balance is not always easy. Part II of this book, "Tuning the Server," addresses this topic in detail.

Tuning the Network

There is really nothing you can do to tune the network *per se*. The network is a resource used by both the clients and the servers. You must make sure that the network does not exceed its limits; other than watching for network overload, the only thing to do is to add more resources (which usually means adding more or faster LAN segments).

Although you cannot do anything to tune the network for more performance, you *can* tune your applications to make good use of the network. Under some operating systems, you can decrease network usage by carefully sizing the network packet size to more closely fit what

SQL*Net wants. You can also reduce network consumption by not sending unnecessary data across the wire.

The important thing is that the network does not cause excessive delays in response time or excessive processing in the clients or servers (caused by exceeding network limitations). If you have a problem, it may be time to upgrade the network.

NOTE: Remember that engineering performance into your system is done at every phase of implementation. Think of performance in the design stage, tune for performance in the test phase, and constantly look for performance problems once the system is in production. The earlier you start thinking about performance, the better off you are in the long run.

Summary

Part I of this book reviewed the Oracle architecture, studied the attributes of a well-tuned system, and looked at the methodology for performance engineering. It also looked at some of the ways you can test performance with benchmarks and how you can monitor the results both through Oracle and third-party products. Finally, you have seen how you can engineer performance into the system at all stages of implementation, from design and test to production.

Chapter 2, “Understanding Terms,” started out by reviewing how Oracle operates, that is, how all its components work together. This review should give you a basis for many of the tuning concepts covered in Part II, “Tuning the Server.”

Chapters 3 and 4 introduced the concept of a well-tuned system and a methodology you can follow to achieve such a system. By understanding the attributes of a good performing system, it may be easier to set goals for your system. Remember that these attributes include good response times as well as high throughput rates. The following parts of the book discuss ways to tune for a specific property such as fast response time or good throughput.

Chapter 5, “Benchmarking,” should have impressed you with the amount of effort involved in creating and running a benchmark. You can use the information in that chapter to help in your next system-purchasing decision. The chapter also looked at some of the attributes of a custom benchmark or workload generator. If you have the opportunity to build a test system, you will see much benefit in both your performance and debugging efforts.

Chapter 6, “Performance Monitoring Tools,” introduced you to some of the products and features of Oracle you may not already be familiar with. Numerous examples in the upcoming chapters use these tools extensively.

Finally, this chapter explained that tuning doesn't have to be left to the test and production phases (when the application is already completed and ready for rollout). Try to influence the design phase. Look for ways to improve performance when the application is still on the drawing board.

I hope that this introduction has given you enough background and review to prepare you for the rest of the book. The following parts of the book look in-depth at some of the performance problems frequently encountered and how to solve them.

Part

Tuning the Server

Chapter 8 What Affects Oracle Server Performance?

- 9** Oracle Instance Tuning
- 10** Performance Enhancements
- 11** Tuning the Server Operating System
- 12** Operating System-Specific Tuning
- 13** System Processors
- 14** Advanced Disk I/O Concepts
- 15** Disk Arrays

Part I of this book reviewed the Oracle architecture and introduced some of the concepts used in the rest of the book. You learned about benchmarks and how you can use benchmarks to compare various architectures. Most importantly, you were given a foundation of guidelines to follow as a tuning methodology.

Part II of this book begins to look at real problems and solutions. Chapter 8, “What Affects Oracle Server Performance?” begins with a discussion about system bottlenecks. You find out what a bottleneck is and how to determine where it is.

Chapter 9, “Oracle Instance Tuning,” and Chapter 10, “Performance Enhancements,” look at tuning specifics such as instance tuning and performance enhancements. These chapters should leave you with a good idea about how to tune the Oracle instance.

Chapter 11, “Tuning the Server Operating System,” looks into what it takes to tune the operating system. The chapter includes specifics about several popular operating systems.

The last three chapters of Part II look at the hardware itself—a concept often missed in tuning books. You learn how the system processors work and how disks operate. Finally, you look at the world of disk arrays: how do they work and how do they help you in an RDBMS environment?

When you finish this part of the book, you should have a pretty good idea about how to tune the server—from configuring the hardware to tuning the instance. Part IV of the book, “Tuning SQL,” contains information about application tuning; but for now, let’s focus on the server.

Chapter 8

What Affects Oracle Server Performance?

Many factors contribute to the performance of Oracle on the server: the tuning of the Oracle instance, the OS, the hardware itself, and the load generated by the users. In tuning the system, you have some control over almost all these factors.

As mentioned in an earlier chapter, I like to break down the performance of a system into three categories: things that are broken, things that need to be optimized, and things that aren't a problem. The first category can cause significant performance loss; the second can cause minor degradation. Sometimes, there is a significant gray area between the first two categories of problems, but the solutions are the same.

Many different situations can cause performance loss. Consider these examples:

- ◆ **An overloaded I/O system.** This can cause the entire system to slow down while waiting for disk requests to return.
- ◆ **Not enough memory.** This can cause additional I/O usage by reducing the cache-hit rate and by causing the operating system to swap or page.
- ◆ **Lack of Oracle resources.** A shared pool that is too small can cause performance problems, as described in Chapter 9, “Oracle Instance Tuning.”
- ◆ **A slow network.** Network performance problems can reduce throughput and cause user response times to rise.

These problems can be caused by any of these parts of the system:

- ◆ **Hardware.** Problems may be caused by defective hardware or by an insufficient amount of some resource such as memory or disk space.
- ◆ **Operating system.** Problems may be caused by tuning issues as well as OS resources used by Oracle.
- ◆ **Oracle.** It is essential to properly tune Oracle for your configuration so that you can realize optimal performance. A poorly tuned Oracle instance can drastically affect performance.

When there is a significant performance loss caused by a limiting factor in the system, this is known as a *bottleneck*.

System Bottlenecks

The term *bottleneck* comes from the shrinking in size of the neck of a bottle. This constriction causes a reduction in flow, limiting the amount of liquid coming out of the bottle. In a similar fashion, this term is used to describe something that is constricting system performance. Over the years, this term has grown to represent any sort of major limiting factor in a computer system.

A bottleneck can significantly reduce the performance of a system while leaving some resources—such as the CPU—completely underutilized. It is the job of the performance engineer to reduce or eliminate bottlenecks.

Finding the Bottleneck

As you will see in the next few chapters, it is not always easy to determine where a bottleneck is and what is causing it. In some cases, it is almost impossible to determine exactly where the problem is.

If the bottleneck is internal to the operating system or to Oracle itself, there is really nothing you can do about it. The developers of Oracle—as well as developers of most operating systems—are constantly testing new releases to eliminate all the bottlenecks they can.

If the bottleneck is in the hardware, you often have an easier time eliminating it. Solving a hardware limitation usually involves simply adding more disk drives or increasing the amount of RAM in the system. If there is an inherent bottleneck in the system's cache design or memory bus, there usually isn't much you can do about it. Most hardware vendors concentrate on reducing system bottlenecks as quickly as they can. For hardware companies that do significant benchmarking, it is especially important that they have optimal performance.

CAUTION: This is a case where the buyer should beware. Many small “clone vendors” do no work to optimize their hardware for RDBMS performance. You may find that the money you save on hardware is spent buying additional components or systems to make up for the deficiency in the original hardware.

Removing the Bottleneck

How do you get rid of a system bottleneck? It takes a lot of analysis, tuning, and hard work. The next few chapters discuss the particulars of how to get rid of specific bottlenecks.

The goal is to remove the bottleneck as a limiting factor in the performance of your system. In other words, turn the bottleneck into a “non-issue.” If disk I/O is causing a bottleneck, tuning the I/O subsystem—by either redistributing your database or by adding additional capacity—should remove I/O as a limiting factor. Once this is done, I/O is no longer an issue in tuning your system. You have moved the limiting factor somewhere else.

System Tuning

How well the system is tuned is also a major factor in the overall performance. For the most part, tuning the system involves resource allocation. By adding or removing a resource in the RDBMS or the OS—or even the hardware—performance can be drastically affected.

The next few sections briefly introduce some of the tuning factors that can alter system performance. Chapters 9 through 15 detail these topics.

Tuning RDBMS Resources

The RDBMS itself takes up many system resources; it uses these resources to serve the user requests. If Oracle does not have enough of a particular resource itself, you can see vast reductions in performance.

Memory is of particular importance to Oracle. If Oracle does not have enough memory resources allocated for the block buffer cache or the data dictionary cache or even the log buffer, Oracle can be out of tune. If enough memory isn't allocated for the buffer cache or the data dictionary cache, you may see very low cache-hit rates. If enough memory isn't allocated for the log buffer, you may experience inefficient access to the redo log.

Increasing the value of some of the Oracle resources such as the buffer cache, the shared pool, and the log buffer can improve cache-hit rates and thus reduce system I/O, resulting in improved performance. Careful analysis should be done to ensure that you are increasing the correct resource the right amount, as described in Chapter 9, “Oracle Instance Tuning.”

You can use other Oracle tuning parameters to take advantage of special operating system features. For example, in the UNIX operating system, you have to tell Oracle that you have Asynchronous I/O (AIO) available—and you have to tell Oracle to use it. Chapter 12, “Operating System-Specific Tuning,” looks at some of these features.

Tuning OS Resources

The operating system is typically tuned to allow Oracle to allocate needed resources. Many operating systems have limitations on how much memory or CPU time a single process or user can consume. Because Oracle is not a regular user—it is a server process—these resources must be increased to allow Oracle to consume vast system resources.

You may have to turn on other operating system features to allow Oracle to use them. Some features such as AIO (mentioned in the preceding section) are not turned on by default in some operating systems. Other new features are just becoming available. Many of these features are discussed in Chapter 12, “Operating System-Specific Tuning.”

Sometimes, you must reduce operating system parameters to make room for Oracle to get resources. For example, you may have to reduce the OS file system buffers so that you can allocate more memory to Oracle (remember that Oracle bypasses the file system buffers).

Tuning Hardware Resources

Hardware cannot be tuned in the same manner as the RDBMS and the OS. Although you can think of hardware tuning as more of a resource balancing act than a tuning exercise, the methodology is the same.

After analyzing the system, you may discover that it is necessary to add hardware resources such as more disks, disk controllers, RAM, cache, CPUs, and so on. Adding hardware resources is all part of the tuning game and requires the same analysis and testing as RDBMS and OS tuning.

Other Tuning Factors

There may yet be other factors limiting the performance of your system, such as inefficiencies in the application code that cause unnecessary table scans or network limitations.

Other processes running concurrently on the system may be stealing badly needed CPU cycles from Oracle. These processes may be using memory that could be allocated for Oracle; they could be causing contention on the same disk devices Oracle is using.

When analyzing the performance of your system, try not to focus so much on the RDBMS that you miss something external to it. Many factors contribute to the database's performance—and these factors may not all be within the database itself.

System Limitations

Finally, the performance of the system is limited by how fast the system can go. Even with the most optimally tuned system, you are limited by the speed of the system's CPUs. In this case, you have done everything possible to reduce bottlenecks; the system performance relies on the speed of the CPUs, as it should.

When you are limited by how fast the computer runs, your job as a performance engineer is done. If you still need more performance, you need more CPUs or more systems (perhaps clusters). Many factors make it unwise to constantly run your system at its performance limits; it is better to leave a little capacity available for peak activity.

Summary

This chapter looked at what affects Oracle performance. Bottlenecks hurt performance in a big way. If these bottlenecks are not internal to the RDBMS, OS, or hardware, you can eliminate them and greatly improve performance.

You can do much in the way of system tuning to help performance by allocating the proper resources to your RDBMS. In some cases, you may benefit from a larger data dictionary cache; in other cases, you may benefit from more block buffers. Chapter 9, “Oracle Instance Tuning,” looks at these things specifically.

You should realize that there is only so much power in a system and that eventually you will reach that limitation. Once you exceed the capacity of the system and have done all possible optimization, it is time to move on and upgrade.

The next few chapters look at these concepts in detail, starting with the Oracle instance, continuing with the operating system, and then dealing with the hardware.

Chapter 9

Oracle Instance Tuning

This chapter looks at the various areas of the Oracle instance (such as memory usage and I/O) that you can affect and what you can change. To effectively tune the instance, you must know what is affected by your changes. This chapter looks at some specific areas within the instance, describes how to know if you have a problem, and explains how to make a change.

NOTE: Oracle instance tuning should be done in conjunction with application tuning. Because much of the instance tuning is based on the data access patterns and memory usage of the application, changes to the application may result in the opportunity to retune the instance for greater optimization. A perfectly optimized Oracle instance cannot make up for a poorly tuned application.

In keeping with the philosophy of organized and effective tuning, I recommend that you make changes one at a time and carefully record the results. As deadlines approach, it is not always possible to extend the testing effort by changing only one parameter at a time. In the long run, however, this approach is much more effective. It is always possible that multiple changes cancel each other out, incorrectly indicating that nothing has happened.

Of course, I am not advocating that you start from scratch every time. Once you have tuned several systems, take the initialization parameters you have found to work and use them as a starting point. From there, you are well on your way to tuning this particular Oracle instance.

This chapter looks first at memory usage within the Oracle instance and then looks at I/O utilization. These two areas are probably where most bottlenecks occur and are the areas you can most effectively tune.

In a computer system, the fastest storage component is the CPU cache, followed by the system memory. I/O to disk is thousands of times slower than an access to memory. This fact is the key for why you try to make effective use of memory whenever possible and defer I/Os whenever you can. The majority of the user response time is actually spent waiting for a disk I/O to occur.

By making good use of caches in memory and reducing I/O overhead, you can optimize performance. The goal is to retrieve data from memory whenever you can and to use the CPU for other activities whenever you have to wait for I/Os. This chapter examines ways to optimize the performance of the system by taking advantage of caching and effective use of the system's CPUs. It first looks at memory, then at I/O, and then at internal Oracle processes.

Tuning Memory

In the Oracle instance, data is stored in two places: in memory and on disk. Memory has by far the best performance but also has the highest cost. Disk, on the other hand, can store vast amounts of data very cost effectively but has very slow performance relative to memory.

Because memory is the better performer, it is desirable to use memory to access data whenever possible. But because of the vast amounts of data usually accessed and the number of users who need this data, there is a lot of contention on this resource. To make most effective use of memory, you must achieve a balance between the memory used for Oracle caching and the memory needed by the users.

Tuning memory in the Oracle instance involves tuning several major areas:

- ◆ The OS (to provide these resources)
- ◆ Private SQL and PL/SQL areas
- ◆ The shared pool
- ◆ The redo log buffer
- ◆ The buffer cache

Tuning the Operating System

The first step in tuning memory for the Oracle instance is to make sure that there are sufficient resources available in the operating system. You cannot allocate memory to Oracle that doesn't exist. Giving Oracle additional memory at the expense of causing paging or swapping is ineffective and hurts performance. Chapter 12, "Operating System-Specific Tuning," looks at OS-specific issues.

The goal of tuning the operating system for Oracle is to provide enough resources so that Oracle can function. The operating system must provide for the following:

- ◆ **Enough memory for the SGA to fit into main memory.** In most operating systems, this involves allocating a special type of memory structure called *shared memory*. Shared memory is provided by the operating system to allow multiple processes to access the same memory through special system calls. In many operating systems, the shared memory is locked into place and cannot be swapped or paged.
- ◆ **Enough memory for the user processes to fit into main memory.** Remember that each shadow process, or dispatcher, also consumes memory. For user processes, the amount of memory consumed depends on the number of users connected.
- ◆ **Avoid paging and swapping.** Although it is not uncommon for some paging or swapping to occur, if it occurs frequently, you should take steps to reduce it.
- ◆ **Enough memory for operating system activities.** Remember that other OS activities may become active at various times and therefore allocate memory.
- ◆ **Enough memory to accommodate Oracle operations such as archiving, loading, online backup, and so on.**

Chapter 12 looks at some of the operating system-specific features available for monitoring the amount of memory available and how to allocate more memory for Oracle. In the OS itself, the main function of tuning memory is to allocate it for Oracle. As you see in the following section, Oracle can take advantage of this memory in several ways.

Tuning the Private SQL and PL/SQL Areas

A *private SQL area* is an area in memory that contains binding information and runtime buffers. Every session that issues SQL statements has a private SQL area; reducing these resources can be very effective when large numbers of users are involved. A private SQL area is further segmented into a persistent area and a runtime area.

The *persistent area* contains binding information used during the query for data input and retrieval. The size of this area depends on the number of binds and the number of columns specified in the statement.

The *runtime area* contains information used while the SQL statement is being executed. The size of the runtime area depends on the complexity and type of SQL statements being issued and the size of the rows being processed. The runtime area is freed after the statement has been executed. In a query operation, the runtime area is freed only after all the rows have been fetched.

In a dedicated server, the private SQL area is located in the user's PGA. In the case of a multithreaded server, the persistent areas and (for SELECT statements) the runtime areas are kept in the SGA.

It is important to make sure that each user can allocate enough memory for his or her private SQL area; in the case of the multithreaded server, there must be sufficient space in the SGA to connect the required number of users.

Cursors can be used in precompilers to improve performance by reducing the frequency of parsing. To take advantage of cursors, it may be necessary to increase the value of the Oracle parameter `OPEN_CURSORS`.

The size of memory needed for each user is determined by the application and tunables such as `OPEN_CURSORS`. This memory is allocated automatically. In some operating systems, the amount of memory that can be allocated per user is controlled by system parameters. If your application or tuning changes the amount of memory for each user process, you may have to increase the value of those OS parameters. You may also have to increase the amount of system RAM or reduce the Oracle instance memory usage to avoid swapping or paging.

Tuning the Shared Pool

To tune the shared pool, you must look at the individual parts of the shared pool. The *shared pool* contains both the library cache and the data dictionary cache. In a multithreaded server, the shared pool is also used to store session information.

Library Cache

The *library cache* contains the shared SQL and PL/SQL areas. Performance can be improved by both increasing the cache-hit rate in the library cache and by speeding access to the library cache by holding infrequently used SQL statements in cache longer.

A cache miss in the shared SQL area occurs when either a parse statement is called and the already parsed statement does not already exist in the shared SQL area or when an application tries to execute an SQL statement and the shared SQL area containing the parsed statement has been deallocated from the library cache.

For an SQL statement to take advantage of SQL or PL/SQL statements that may have already been parsed, the following criteria must be met:

- ◆ The text of the SQL statement must be identical to the SQL statement that has already been parsed. This includes whitespaces.
- ◆ References to schema objects in the SQL statements must resolve to the same object.
- ◆ Bind variables must match the same name and data type.
- ◆ The SQL statements must be optimized using the same approach; in the case of the cost-base approach, the same optimization goal must be used.

At first glance, you may think that many of these conditions make it difficult to take advantage of the shared SQL areas. But users sharing the same application code can easily take advantage of already parsed shared SQL statements. It is to the advantage of the application developer to use the same SQL statements to access the same data and thus ensure that SQL statements within the application can also take advantage of this caching.

Using stored procedures whenever possible guarantees that the same shared PL/SQL area is used. Another advantage is that stored procedures are stored in a parsed form, eliminating runtime parsing altogether.

TIP: Standardizing on naming conventions for bind variables and on spacing conventions for SQL and PL/SQL statements also increases the likelihood of reusing shared SQL statements.

The V\$LIBRARYCACHE table contains statistics on how well you are utilizing the library cache. The important columns to view in this table are PINS and RELOADS.

- ◆ *PINS*: The number of times the item in the library cache was executed.
- ◆ *RELOADS*: The number of times the library cache misses and the library object must be reloaded.

A low number of reloads relative to the number of executions indicates a high cache-hit rate. To get an idea of the total number of cache misses, use this statement:

```
SQL> SELECT SUM(reloads) "Cache Misses",
  2  SUM(pins) "Executions",
  3  100 * ( SUM(reloads) / SUM(pins) ) "Cache Miss Percent"
  4  FROM v$librarycache;
```

Cache Misses	Executions	Cache Miss Percent
9	2017	.44620724

The sample output shown here indicates that a total of 2,017 SQL statements, PL/SQL blocks, and object definitions were accessed with only 9 having to reload because they had aged out of the library cache. This means that only .44 percent of these statements resulted in reparsing.

To look at the cache hits based on the types of statements, you can use the following statement:

```
SQL> SELECT namespace,
  2  reloads "Cache Misses",
  3  pins "Executions"
  4  FROM v$librarycache;
```

NAMESPACE	Cache Misses	Executions
SQL AREA	4	1676
TABLE/PROCEDURE	5	309
BODY	0	0
TRIGGER	0	0
INDEX	0	21
CLUSTER	0	15
OBJECT	0	0
PIPE	0	0

8 rows selected.

The total amount of reloads should be near zero. If you see more than 1 percent library cache misses, take action. You can reduce the number of cache misses by writing identical SQL statements or by increasing the size of the library cache.

You should be able to reduce the library cache misses by increasing the amount of memory available for the library cache. This can be done by increasing the Oracle tunable parameter `SHARED_POOL_SIZE`.

You may also have to increase the number of cursors available for a session by increasing the Oracle parameter `OPEN_CURSORS`.

Be careful not to increase the amount of memory you require beyond that set aside by the operating system. Any paging or swapping caused by over-allocating memory will offset any advantage you get from the library cache.

If you have plenty of memory, you may be able to speed access to the shared SQL areas by setting the Oracle initialization parameter `CURSOR_SPACE_FOR_TIME` equal to TRUE. When this parameter is TRUE, it specifies that a shared SQL area cannot be deallocated until all the cursors associated with it are closed.

If `CURSOR_SPACE_FOR_TIME` is TRUE, it is not necessary for Oracle to check whether the SQL statement is in the library cache because this parsed statement cannot be deallocated as long as the cursor is open. If memory is scarce on your system, do not set this parameter. If the value is TRUE and there is no space in the shared pool for a new SQL statement, an error will be returned, thus halting the application.

Data Dictionary Cache

The *data dictionary* contains a set of tables and views Oracle uses as a reference to the database. Oracle stores information here about both the logical and physical structure of the database. The data dictionary contains information such as the following:

- ◆ User information such as user privileges.
- ◆ Integrity constraints defined for tables in the database.
- ◆ Names and data types of all columns in database tables.
- ◆ Information on space allocated and used for schema objects.

The data dictionary is frequently accessed by Oracle itself for the parsing of SQL statements. This access is essential to the operation of Oracle; performance bottlenecks in the data dictionary affect all Oracle users.

You can check the efficiency of the data dictionary cache. Statistics for the data dictionary cache are stored in the dynamic performance table V\$ROWCACHE (the data dictionary cache is sometimes known as the *row cache*). The important columns to view in this table are GETS and GETMISSES:

- ◆ *GETS*: The total number of requests for the particular item.
- ◆ *GETMISSES*: The total number of requests resulting in cache misses.

A few number of cache misses are expected, especially during startup when the cache has not been populated. To get an idea of the total number of cache misses, use the following statement:

```
SQL> SELECT SUM(getmisses) "Cache Misses",
  2  SUM(gets) "Requests",
  3  100 * ( SUM(getmisses) / SUM(gets) ) "Cache Miss Percent"
  4  FROM v$rowcache;
```

Cache Misses	Requests	Cache Miss Percent
277	2185	12.677346

The output shown here indicates that a sum of 2,185 requests were made to the data dictionary cache with 277 data dictionary cache misses. This means that 12.68 percent of these requests caused a cache miss.

To get the cache miss statistics on the individual elements of the data dictionary, execute this SQL statement:

```
SQL> SELECT parameter,
  2  getmisses "Cache Misses",
  3  gets "Requests"
  4  FROM v$rowcache;
```

PARAMETER	Cache Misses	Requests
dc_free_extents	4	92
dc_used_extents	0	0
dc_segments	1	7
dc tablespaces	0	0
dc tablespaces	0	0
dc_tablespace_quotas	0	0
dc_files	0	0
dc_users	8	90
dc_rollback_segments	4	122
dc_objects	44	378
dc_constraints	0	0
dc_object_ids	0	0
dc_tables	20	318
dc_synonyms	5	9
dc_sequences	1	6
dc_usernames	4	71
dc_database_links	0	0
dc_histogram_defs	0	0
dc_profiles	0	0
dc_users	0	0
dc_columns	153	1028
PARAMETER	Cache Misses	Requests
dc_table_grants	14	18
dc_column_grants	0	0
dc_indexes	12	176
dc_constraint_defs	7	7
dc_constraint_defs	0	0
dc_sequence_grants	0	0
dc_user_grants	7	62

28 rows selected.

In frequently accessed dictionary caches, the miss rate should not rise above 10 to 15 percent. If the percent of misses continues to increase during run time, increase the amount of memory allocated for the data dictionary cache. Use the same parameter as for the library cache: SHARED_POOL_SIZE.

Shared Session

In the multithreaded server configuration, the session information is also stored in the shared pool. This information includes the private SQL areas as well as sort areas. It is important to make sure that you do not run out of space in the shared pool for this information.

To determine whether you should increase space for these shared sessions, you can extract the sum of memory allocated for all sessions and the maximum amount of memory allocated for sessions from the dynamic performance table V\$SESSTAT. The information in this table is split into the memory used by the UGA (User Global Area) and that used by the PGA (Program Global Area). To retrieve this information, use a query such as the one in Listing 9.1.

Listing 9.1 Retrieving Information from V\$SESSTAT

```

select sid, value "Session UGA Memory"
from v$sesstat,v$statname
WHERE name = 'session uga memory' AND
v$sesstat.statistic# = v$statname.statistic#;

select SUM(value) "Sum of Session UGA Memory"
from v$sesstat,v$statname
WHERE name = 'session uga memory' AND
v$sesstat.statistic# = v$statname.statistic#;

select sid, value "Session PGA Memory"
from v$sesstat,v$statname
WHERE name = 'session pga memory' AND
v$sesstat.statistic# = v$statname.statistic#;

select SUM(value) "Sum of Session PGA Memory"
from v$sesstat,v$statname
WHERE name = 'session pga memory' AND
v$sesstat.statistic# = v$statname.statistic#;

select sid, value "Session UGA Memory Max"
from v$sesstat,v$statname
WHERE name = 'session uga memory max' AND
v$sesstat.statistic# = v$statname.statistic#;

select SUM(value) "Sum of Session UGA Memory Max"
from v$sesstat,v$statname
WHERE name = 'session uga memory max' AND
v$sesstat.statistic# = v$statname.statistic#;

select sid, value "Session PGA Memory Max"
from v$sesstat,v$statname
WHERE name = 'session pga memory max' AND
v$sesstat.statistic# = v$statname.statistic#;

select SUM(value) "Sum of Session PGA Memory Max"
from v$sesstat,v$statname
WHERE name = 'session pga memory max' AND
v$sesstat.statistic# = v$statname.statistic#;

```

The result of this query looks something like Listing 9.2.

Listing 9.2 The Results of the Query in Listing 9.1

SID	Session UGA Memory
1	7048
54	43928
5	14104
4	7048
3	7048
2	8956

6 rows selected.

continues

Listing 9.2 continued

Sum of Session UGA Memory

83896

SID Session PGA Memory

1 34032
54 76592
5 51056
2 42544
4 42544
3 42544

6 rows selected.

Sum of Session PGA Memory

289312

SID Session UGA Memory Max

1 7048
3 7048
5 18288
54 48168
4 7048
2 8956

6 rows selected.

Sum of Session UGA Memory Max

96556

SID Session PGA Memory Max

1 34032
2 42544
4 42544
54 76592
5 51056
3 42544

6 rows selected.

Sum of Session PGA Memory Max

289312

The SID represents the session ID, which shows you the value for each server session.

Tuning the Buffer Cache

Probably the most important Oracle cache in the system is the buffer cache. The *buffer cache* makes up the majority of the Oracle SGA and is used for every query and update in the system.

Each time a data block is read, it is copied into the buffer cache. Each time a modification is made, it is done in the buffer cache. Remember that each server process accesses the buffer cache directly.

In a read operation, the server process first checks to see whether the requested data is already in the SGA. If it is, the data is accessed directly from the SGA. If the data is not already in the SGA, the server process copies the data from the data file into the SGA where it will be accessed.

In an update operation, the server process modifies the data block buffer(s) in the SGA only. It is up to the DBWR to write these dirty buffers out to disk. With the exception of the CKPT process, only the DBWR writes to the data files.

Because the buffer cache is accessed so frequently, it is important that the buffer cache has sufficient size to get a good cache-hit rate. The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The important columns to view in this table are as follows:

- ◆ *PHYSICAL READS*: This is the total number of requests that result in a disk access; this is a *cache miss*.
- ◆ *DB BLOCK GETS* and *CONSISTENT GETS*: The sum of the two values DB BLOCK GETS and CONSISTENT GETS represents the total number of requests for data.

To see how well the block buffer cache is doing, use this query:

```
SQL> SELECT name, value
  2  FROM v$sysstat
  3 WHERE name IN ('db block gets', 'consistent gets', 'physical reads');
```

NAME	VALUE
db block gets	155
consistent gets	5293
physical reads	334

To calculate the cache hit ratio, use this formula:

```
Cache Hit Ratio = 1 - ( PHYSICAL READS / ( DB BLOCK GETS + CONSISTENT GETS ))
Cache Hit Ratio = 1 - (334 / ( 155 + 5293 ) ) = 1 - (334 / 5448 ) = 1 - 0.0613 = 0.938
```

This example shows a cache hit rate of 93.8 percent.

If the cache-hit rate is lower than 70 or 80 percent, you may need to increase the database buffer cache to improve performance. The buffer cache can be increased by tuning the Oracle initialization parameter `DB_BLOCK_BUFFERS`.

The initialization parameter `DB_BLOCK_BUFFERS` specifies the number of database block buffers configured in the system. The total amount of space utilized by these buffers is calculated as follows:

```
Buffer Size = DB_BLOCK_BUFFERS * DB_BLOCK_SIZE
```

CAUTION: Be careful not to configure `DB_BLOCK_BUFFERS` to use more memory than the system has allocated. Doing so can cause swapping or paging to occur in some systems or a failure of the Oracle instance to start up in other systems.

You can obtain an estimate of additional cache hits that would be achieved with more database block buffers through the virtual table `X$KCBRBH`. When this table is enabled through the Oracle initialization parameter `DB_BLOCK_LRU_EXTENDED_STATISTICS`, it contains estimates of the performance of a larger buffer cache.

The parameter `DB_BLOCK_LRU_EXTENDED_STATISTICS` specifies the number of rows in the `X$KCBRBH` table. Each row contains the estimated cache hits obtained by those additional buffers. The table `X$KCBRBH` contains the following columns:

- ◆ *INDX*: The value is 1 less than the number of buffers that would be added.
- ◆ *COUNT*: The value is the estimated number of additional cache hits that would be obtained by adding *INDX* + 1 buffers.

Be sure to run your application while you are gathering these statistics. Doing so gives you a better representation of how your particular system will react with more database block buffers.

Because the data files and the database block buffers make up the majority of the I/Os in the system, it is important to have a sufficiently large buffer cache. Don't increase your database block buffers at the expense of the shared pool. Although the shared pool is not as large or as heavily used, it serves a critical purpose in the execution of SQL statements.

Any time you make a significant change in `DB_BLOCK_BUFFERS`—or any Oracle parameter—go back and check the OS for different I/O rates. Also check within Oracle to see whether the cache-hit rates have changed, as well as the I/O rates.

Tuning the I/O Subsystem

I/O is probably one of the most common problems facing Oracle users. In many cases, the performance of the system is entirely limited by disk I/O. In some cases, the system actually becomes idle waiting for disk requests to complete. We say that these systems are *I/O bound* or *disk bound*.

As you see in Chapter 14, “Advanced Disk I/O Concepts,” disks have certain inherent limitations that cannot be overcome. Therefore, the way to deal with disk I/O issues is to understand the limitations of the disks and design your system with these limitations in mind. Knowing the performance characteristics of your disks can help you in the design stage.

Optimizing your system for I/O should happen during the design stage. As you see in Part III, “Configuring the System,” different types of systems have different I/O patterns and require different I/O designs. Once the system is built, you should first tune for memory and then tune for disk I/O. The reason you tune in this order is to make sure that you are not dealing with excessive cache misses, which cause additional I/Os.

The strategy for tuning disk I/O is to keep all drives within their physical limits. Doing so reduces queuing time—and thus increases performance. In your system, you may find that some disks process many more I/Os per second than other disks. These disks are called “hot spots.” Try to reduce hot spots whenever possible. Hot spots occur whenever there is a lot of contention on a single disk or set of disks.

Understanding Disk Contention

Disk contention occurs whenever the physical limitations of a disk drive are reached and other processes have to wait. Disk drives are mechanical and have a physical limitation on both disk seeks per second and throughput. If you exceed these limitations, you have no choice but to wait.

You can find out if you are exceeding these limits both through Oracle’s file I/O statistics and through operating system statistics. This chapter looks at the Oracle statistics; Chapter 12, “Operating System-Specific Tuning,” looks at the operating system statistics for some popular systems.

Although the Oracle statistics give you an accurate picture of how many I/Os have taken place for a particular data file, they may not accurately represent the entire disk because other activity outside of Oracle may be incurring disk I/Os. Remember that you must correlate the Oracle data file to the physical disk on which it resides.

Information about disk accesses is kept in the dynamic performance table V\$FILESTAT. Important information in this table is listed in the following columns:

- ◆ *PHYRDS*: The number of physical reads done to the data file.
- ◆ *PHYWRTS*: The number of physical writes done to the data file.

The information in V\$FILESTAT is referenced by file number. The dynamic performance table V\$DATAFILE contains a reference to this number as well as other useful information such as this:

- ◆ *NAME*: The name of the data file.
- ◆ *STATUS*: The type of file and its current status.
- ◆ *BYTES*: The size of the data file.

Together, the V\$FILESTAT and V\$DATAFILE tables can give you an idea of the I/O usage of your data files. Use the following query to get this information:

```
SQL> SELECT substr(name,1,40), phyrd, phywrts, status, bytes
  2  FROM v$datafile df, v$filestat fs
  3 WHERE df.file# = fs.file#;
```

SUBSTR(NAME,1,40)	PHYRDS	PHYWRTS	STATUS	BYTES
C:\UTIL\ORAWIN\DBS\wdbsys.ora	221	7	SYSTEM	10485760
C:\UTIL\ORAWIN\DBS\wdbuser.ora	0	0	ONLINE	3145728
C:\UTIL\ORAWIN\DBS\wdbrbs.ora	2	0	ONLINE	3145728
C:\UTIL\ORAWIN\DBS\wdbtemp.ora	0	0	ONLINE	2097152

The total I/O for each data file is the sum of the physical reads and physical writes. It is important to make sure that these I/Os don't exceed the physical limitations of any one disk. I/O throughput problems to one disk may slow down the entire system depending on what data is on that disk. It is particularly important to make sure that I/O rates are not exceeded on the disk drives.

Identifying Disk Contention Problems

To identify disk contention problems, you must analyze the I/O rates of each disk drive in the system. If you are using individual disks or disk arrays, the analysis process is slightly different.

For individual disk drives, simply invoke your operating system or third-party tools and check the number of I/Os per second on an individual disk basis. This process gives you an accurate representation of the I/O rates on each drive. As you learn in Chapter 14, "Advanced Disk I/O Concepts," a general rule of thumb is not to exceed 50 I/Os per second per drive with random access, or 100 I/Os per second per drive with sequential access. If you are experiencing a disk I/O problem, you may see excessive idle CPU cycles and poor response times.

For a disk array, also invoke your operating system or third-party tools and check the same items—specifically the number of I/Os per second per disk. The entire disk array appears as one disk. For most popular disk arrays on the market today, it is accurate to simply divide the I/O rate by the number of disks to get the I/Os per second per disk rate.

NOTE: Although the Oracle dynamic performance tables can be quite useful in telling you how many physical I/Os are generated for each individual data file, they cannot accurately report I/O rates. Oracle only reports the number of I/Os—not the rate at which they occur. Oracle also does not report any disk activity outside the Oracle instance.

The next step in identifying a disk contention problem is to determine the I/O profile for your disk. It is sufficient to split this into two major categories: sequential and random I/O. Here is what to look for:

- ◆ **Sequential I/O.** In sequential I/O, data is written or read from the disk in order, so very little head movement occurs. Access to the redo log files is always sequential.
- ◆ **Random I/O.** Random I/O occurs when data is accessed in different places on the disk, causing head movement. Access to data files is almost always random. For database loads, access is sequential; in most other cases (especially OLTP), the access patterns are almost always random.

With sequential I/O, the disk can operate at a much higher rate than it can with random I/O. If *any* random I/O is being done on a disk, the disk is considered to be accessed in a random fashion. Even if you have two separate processes that access data in a sequential manner, the I/O pattern is random (refer to Chapter 14, “Advanced Disk I/O Concepts”).

With random I/O, there is not only access to the disk but a large amount of head movement, which reduces the performance of the disks. Chapter 14 contains an in-depth explanation of why this is true.

Finally, check these rates against the recommended I/O rates for your disk drives. Here are some good guidelines:

- ◆ **Sequential I/O.** A typical SCSI-II disk drive can support approximately 100 to 150 sequential I/Os per second.
- ◆ **Random I/O.** A typical SCSI-II disk drive can support approximately 50 to 60 random I/Os per second.

Chapter 14 shows how these sequential and random I/O rates are determined and how you can recalculate them for your particular disks.

Solving Disk Contention Problems

There are a few rules of thumb you should follow in solving disk contention problems:

- ◆ **Isolate sequential I/Os.** Because sequential I/Os can occur at a much higher rate, isolating them lets you run these drives much faster.
- ◆ **Spread out random I/Os as much as possible.** You can do this by striping table data through Oracle striping, OS striping, or hardware striping.
- ◆ **Separate data and indexes.** By separating a heavily used table from its index, you allow a query to a table to access data and indexes on separate disks simultaneously.
- ◆ **Eliminate non-Oracle disk I/O from disks that contain database files.** Any other disk I/O slow down Oracle access to these disks.

The following sections look at each of these solutions and determine how they can be accomplished.

Isolate Sequential I/Os

Isolating sequential I/Os allows you to drive sequentially accessed disks at a much higher rate than randomly accessed disks. Isolating sequential I/Os can be accomplished by simply putting the Oracle redo log files on separate disks.

Be sure to put each redo log file on its own disk—especially the mirrored log file (if you are mirroring with Oracle). If you are mirroring with OS or hardware mirroring, the redo log files will already be on separate volumes. Although each log file is written sequentially, having the mirror on the same disk causes the disk to seek between the two log files between writes, thus degrading performance.

CAUTION: A common mistake is to put multiple redo log files on the same disk. Although each log file is written sequentially, once a log switch occurs and the next log in the sequence is being written to, the ARCH processes start archiving the previous log. This causes two sequential processes to access data sequentially to the same disk, thus randomizing the access.

It is important to protect redo log files against system failures by mirroring them. You can do this through Oracle itself, or by using OS or hardware fault tolerance features.

Spread Out Random I/Os

By the very nature of random I/Os, accesses are to vastly different places in the Oracle data files. This pattern makes it easy for random I/O problems to be alleviated by simply adding more disks to the system and spreading the Oracle tables across these disks. You can do this by striping the data across multiple drives or (depending on your configuration) by simply putting tables on different drives.

Striping is the act of transparently dividing the contents of a large data source into smaller sources. Striping can be done through Oracle, the OS, or through hardware disk arrays.

Oracle Striping

Oracle striping involves dividing a table's data into small pieces and further dividing these pieces among different data files. Oracle striping is done at the tablespace level with the `CREATE TABLESPACE` command. To create a striped tablespace, use a command similar to this one:

```
SQL> CREATE TABLESPACE mytablespace
  2  DATAFILE 'file1.dbf' SIZE 500K,
  3          'file2.dbf' SIZE 500K,
  4          'file3.dbf' SIZE 500K,
  5          'file4.dbf' SIZE 500K;
```

```
Tablespace created.
```

To complete this task, you must then create a table within this tablespace with four extents. This creates the table across all four data files, which (hopefully) are each on their own disk. Create the table with a command like this one:

```
SQL> CREATE TABLE mytable
  2  ( name varchar(40),
  3    title varchar(20),
  4    office_number number(4) )
  5 TABLESPACE mytablespace
  6 STORAGE ( INITIAL 495K NEXT 495K
  7  MINEXTENTS 4 PCTINCREASE 0 );
```

Table created.

In this example, each data file has a size of 500K. This is called the *stripe size*. If the table is large, the stripes are also large (unless you add many stripes). Large stripes can be an advantage when you have large pieces of data within the table, such as BLOBs. In most OLTP applications, it is more advantageous to have a smaller striping factor to distribute the I/Os more evenly.

The size of the data files depends on the size of your tables. Because it is difficult to manage hundreds of data files, it is not uncommon to have one data file per disk volume per tablespace. If your database is 10 gigabytes in size and you have 10 disk volumes, your data file size will be 1 gigabyte.

When you add more data files of a smaller size, your I/Os are distributed more evenly, but the system is harder to manage because there are more files. As you see in Chapter 15, “Disk Arrays,” you can achieve both manageability and ease of use by using a hardware or software disk array.

Oracle striping can be used in conjunction with OS or hardware striping.

OS Striping

Depending on the operating system, striping can be done at the OS level either through an operating system facility or through a third-party application. OS striping is done at OS installation time.

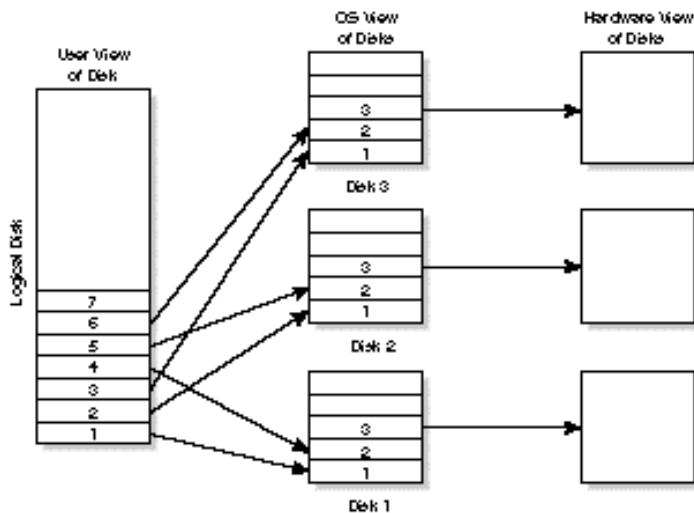
OS disk striping is done by taking two or more disks and creating one large *logical disk*. In sequence, the stripes appear on the first disk, then the second disk, and so on (see Figure 9.1). The size of each stripe depends on the OS and the striping software you are running.

To figure out which disk has the desired piece of data, the OS must keep track of where the data is. To do this, a certain amount of CPU time must be spent maintaining this information. If fault tolerance is used, even more CPU resources are required.

Depending on the software you are using to stripe the disks, the OS monitoring facilities may display disk I/O rates on a per-disk basis or on a per-logical-disk basis. Regardless of how the information is shown, you can easily determine the I/O rate per disk.

Figure 9.1

Operating system striping.



Many of the OS-striping software packages on the market today can also take advantage of RAID technology to provide a measure of fault tolerance. OS striping is very good; however, it does consume system resources that hardware striping does not.

RAID Technology

The various RAID (Redundant Array of Inexpensive Disks) levels are discussed in detail in Chapter 15, “Disk Arrays,” but here is a brief synopsis:

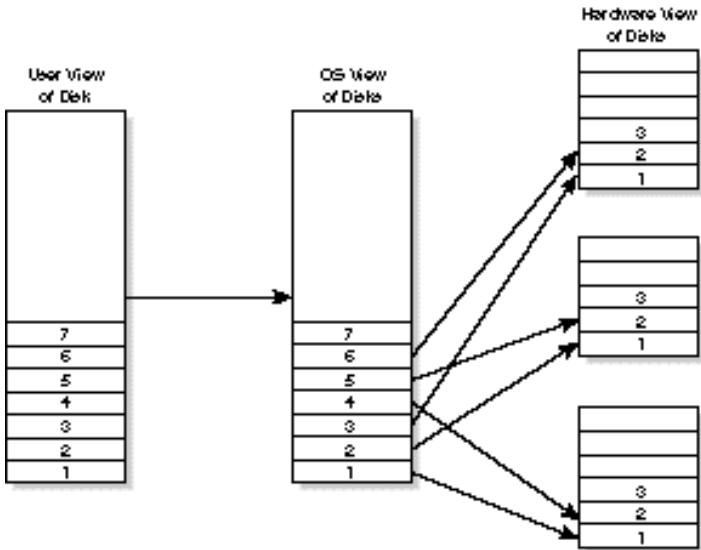
- ◆ RAID 0. Sometimes known as No Fault Tolerance (NFT). Data is striped across the disks.
- ◆ RAID 1. Mirroring. Data is duplicated across sets of drives. This level can be used in conjunction with RAID 0 to create striped and mirrored volumes.
- ◆ RAID 2. Bit-level striping. This level of striping is theoretical and is not used in practice.
- ◆ RAID 3. Sector-level striping. This level of striping is not usually seen in practice.
- ◆ RAID 4. Parity disk. One disk in four is used as a parity drive for fault tolerance. The data is also striped as in RAID 0.
- ◆ RAID 5. Distributed parity. Parity is distributed across many disks. The size of the volume is $(n - 1) * \text{Disk Size}$ where n is the number of drives in the set.

Hardware Striping

Hardware striping has a similar effect to OS striping. Hardware fault tolerance is obtained by replacing your disk controller with a disk array. A *disk array* is a controller that uses many disks

to make one *logical* disk. The system takes a small slice of data from each of the disks in sequence to make up the larger logical disk (see Figure 9.2).

Figure 9.2
Hardware striping.



As you can see in Figures 9.1 and 9.2, to the user and the RDBMS software, the effect is the same whether you use OS or hardware disk striping. The main difference between the two is where the actual overhead of maintaining the disk array is maintained.

Hardware fault tolerance has the advantage of not taking any additional CPU or memory resources on the server. All the logic to do the striping is done at the controller level. As with OS striping, hardware striping can also take advantage of RAID technology for fault tolerance.

Review of Striping Options

Whether you use Oracle striping, OS striping, or hardware striping, the goal is the same: distribute the random I/Os across as many disks as possible. In this way, you can keep the number of I/Os per second requested within the bounds of the physical disks.

If you use Oracle striping or OS striping, you can usually monitor the performance of each disk individually to see how hard they are being driven. If you use hardware striping, remember that the OS monitoring facilities typically see the disk volume as one logical disk. You can easily determine how hard the disks are being driven by dividing the I/O rate by the number of drives.

With hardware and OS striping, the stripes are small enough that the I/Os are usually divided among the drives fairly evenly. Be sure to monitor the drives periodically to verify that you are not up against I/O limits.

Use this formula to calculate the I/O rate per drive:

I/Os per disk = (Number of I/Os per second per volume) / (Number of drives in the volume)

Suppose that you have a disk array with four drives generating 120 I/Os per second. The number of I/Os per second per disk is calculated as follows:

I/Os per disk = 120 / 4 = 30 I/Os per second per disk

For data volumes that are accessed randomly, you don't want to push the disks past 50 to 60 I/Os per second per disk. To estimate how many disks you need for data volumes, use this formula:

Number of disks = I/Os per second needed / 60 I/Os per second per disk

If your application requires a certain data file to supply 500 I/Os per second (based on analysis and calculations), you can estimate the number of disk drives needed as follows:

Number of disks = 500 I/Os per second / 60 I/Os per second per disk = 16 2/3 disks
or 17 disks

This calculation gives you a good approximation for how large to build the data volumes with no fault tolerance. Chapter 15, "Disk Arrays," looks at various RAID levels and additional I/Os that are incurred by adding disk fault tolerance.

Separate Data and Indexes

Another way to reduce disk contention is to separate the data files from their associated indexes. Remember that disk contention is caused by multiple processes trying to obtain the same resources. For a particularly "hot" table with data that many processes try to access, the indexes associated with that data will be "hot" also.

Placing the data files and index files on different disks reduces the contention on particularly hot tables. Distributing the files also allows more concurrency by allowing simultaneous accesses to the data files and the indexes. Look at the Oracle dynamic performance tables to determine which tables and indexes are the most active.

Eliminate Non-Oracle Disk I/Os

Although it is not necessary to eliminate all non-Oracle I/Os, reducing significant I/Os will help performance. Most systems are tuned to handle a specific throughput requirement or response time requirement. Any additional I/Os that slow down Oracle can affect both these requirements.

Another reason to reduce non-Oracle I/Os is to increase the accuracy of the Oracle dynamic performance table, V\$FILESTAT. If only Oracle files are on the disks you are monitoring, the statistics in this table should be very accurate.

Reducing Unnecessary I/O Overhead

Reducing unnecessary I/O overhead can increase the throughput available for user tasks. Unnecessary overhead such as *chaining* and *migrating* of rows hurts performance.

Migrating and chaining occur when an UPDATE statement increases the size of a row so that it no longer fits in the data block. When this happens, Oracle tries to find space for this new row. If a block is available with enough room, Oracle moves the entire row to that new block. This is called *migrating*. If no data block is available with enough space, Oracle splits the row into multiple pieces and stores them in several data blocks. This is called *chaining*.

Migrated and Chained Rows

Migrated rows cause overhead in the system because Oracle must spend the CPU time to find space for the row and then copy the row to the new data block. This takes both CPU time and I/Os. Therefore, any UPDATE statement that causes a migration incurs a performance penalty.

Chained rows cause overhead in the system not only when they are created but each time they are accessed. A chained row requires more than one I/O to read the row. Remember that Oracle reads from the disk data blocks; each time the row is accessed, multiple blocks must be read into the SGA.

You can check for chained rows with the LIST CHAINED ROWS option of the ANALYZE command. You can use these SQL statements to check for chained or migrated rows:

```
SQL> Rem
SQL> CREATE TABLE chained_rows (
 2  owner_name      varchar2(30),
 3  table_name      varchar2(30),
 4  cluster_name    varchar2(30),
 5  head_rowid      rowid,
 6  timestamp        date);

Table created.

SQL> Rem
SQL> Rem Analyze the Table in Question
SQL> Rem
SQL> ANALYZE
 2  TABLE scott.emp LIST CHAINED ROWS;

Table analyzed.

SQL> Rem
SQL> Rem Check the Results
SQL> Rem
SQL> SELECT * from chained_rows;

no rows selected
```

If any rows are selected, you have either chained or migrated rows. To solve the problem of migrated rows, copy the rows in question to a temporary table, delete the rows from the initial table, and reinsert the rows into the original table from the temporary table.

Run the chained-row command again to show only chained rows. If you see an abundance of chained rows, this is an indication that the Oracle database block size is too small. You may want to export the data and rebuild the database with a larger block size.

You may not be able to avoid having chained rows, especially if your table has a LONG column or long CHAR or VARCHAR2 columns. If you are aware of very large columns, it can be advantageous to adjust the database block size before implementing the database.

A properly sized block ensures that the blocks are used efficiently and I/Os are kept to a minimum. Don't over-build the blocks or you may end up wasting space. The block size is determined by the Oracle parameter DB_BLOCK_SIZE. Remember that the amount of memory used for database block buffers is calculated as follows:

```
Memory used = DB_BLOCK_BUFFERS (number) * DB_BLOCK_SIZE (bytes)
```

Be careful to avoid paging or swapping caused by an SGA that doesn't fit into RAM.

Dynamic Extensions

Additional I/O is generated by the extension of segments. Remember that segments are allocated for data in the database at creation time. As the table grows, extents are added to accommodate this growth.

Dynamic extension not only causes additional I/Os, it also causes additional SQL statements to be executed. These additional calls, known as *recursive calls*, as well as the additional I/Os can impact performance.

You can check the number of recursive calls through the dynamic performance table, V\$SYSSTAT. Use the following command:

```
SQL> SELECT name, value
  2  FROM v$sysstat
  3 WHERE name = 'recursive calls';
```

NAME	VALUE
recursive calls	5440

Check for recursive calls after your application has started running and then 15 to 20 minutes later. This information will tell you approximately how many recursive calls the application is causing. Recursive calls are also caused by the following:

- ◆ Execution of Data Definition Language statements.
- ◆ Execution of SQL statements within stored procedures, functions, packages, and anonymous PL/SQL blocks.

- ◆ Enforcement of referential integrity constraints.
- ◆ The firing of database triggers.
- ◆ Misses on the data dictionary cache.

As you can see, many other conditions can also cause recursive calls. One way to check whether you are creating extents dynamically is to check the table DBA_EXTENTS. If you see that many extents have been created, it may be time to export your data, rebuild the tablespace, and reload the data.

Sizing a segment large enough to fit your data properly benefits you in two ways:

- ◆ Blocks in a single extent are contiguous and allow multiblock reads to be more effective, thus reducing I/O.
- ◆ Large extents are less likely to be dynamically extended.

Try to size your segments so that dynamic extension is generally avoided and there is adequate space for growth.

PCTFREE and PCTUSED Command Options

Another way to improve performance and decrease overhead is to use the PCTFREE and PCTUSED options of the STORAGE clause in the CREATE CLUSTER, CREATE TABLE, CREATE INDEX, and CREATE SNAPSHOT commands. By using PCTFREE and PCTUSED, you have more exact control over the use of the data blocks themselves. In many cases, knowing your application and your data can help you improve overall system performance.

You use PCTFREE and PCTUSED for several purposes. Both options are of a performance nature and are space related. PCTFREE and PCTUSED can effectively speed up access to data blocks—but at the price of wasting space if you’re not careful. Another important effect of PCTFREE and PCTUSED is to reduce chaining.

Think of PCTFREE as a *high water mark* and PCTUSED as a *low water mark*. If the space in a data block is such that there is less space left than PCTFREE, no new rows can be added in that block until the amount of space in the table is less than PCTUSED.

The sum of PCTFREE and PCTUSED cannot exceed 100. Because PCTFREE actually represents a high water mark of $100 - \text{PCTFREE}$, if the sum of the two exceeds 100, there is an inconsistency in the formula and PCTFREE would be less than PCTUSED.

An Example

Assume that you have the following values for PCTFREE and PCTUSED:

```
PCTFREE = 20
PCTUSED = 40
```

In this example, you can add new rows to the data block until the data block becomes 80 percent full (100 percent – 20 percent free). When this occurs, no more rows can be added to this data block; the space is reserved for growth of the existing rows.

You can add new rows to the data block only when the percentage of available space in the block has been reduced to 40 percent (used). This effectively saves space in the data blocks for growth of rows and avoids chaining.

The defaults for `PCTFREE` and `PCTUSED` are as follows:

```
PCTFREE = 10  
PCTUSED = 40
```

PCTFREE

`PCTFREE` has the effect of reserving space in the data block for growth of existing rows. New rows can be added to the data block until the amount of space remaining in the data block is less than `PCTFREE` percent.

A high `PCTFREE` value has the following effects:

- ◆ There is a large amount of space for growth of existing rows.
- ◆ Improves performance because blocks have to be reorganized less frequently.
- ◆ Improves performance because chaining is reduced.
- ◆ Typically requires more space because blocks are not used as efficiently. There will always be a moderate amount of empty space in the data blocks.

A lower `PCTFREE` value has the opposite effects:

- ◆ There is less space for growth of existing rows.
- ◆ Performance is reduced because reorganization may become more frequent
- ◆ Performance is reduced because rows may have to be chained more often, increasing CPU use as well as causing additional I/Os.
- ◆ Space is used more effectively. Blocks are filled more completely, thus reducing waste.

Using `PCTFREE` can help if you have an application that frequently inserts new data into rows. Because the `PCTFREE` option is used in the `CREATE CLUSTER`, `CREATE TABLE`, `CREATE INDEX`, and `CREATE SNAPSHOT` commands, it is worth the effort to look at each of your tables, clusters, and indexes individually and decide on an effective `PCTFREE` value for each.

PCTUSED

Once `PCTFREE` is reached, no new rows can be inserted into the data block until the space in the block has fallen below `PCTUSED`. Another feature of `PCTUSED` is that Oracle tries to keep a data block at least `PCTUSED` full before using new blocks.

A high PCTUSED value has the following effects:

- ◆ Decreases performance because you usually have more migrating and chained rows.
- ◆ Reduces space waste by filling the data block more completely.

A lower value for PCTUSED has the following effects:

- ◆ Performance is improved because of the drop in the number of migrated and chained rows.
- ◆ Less efficient space usage caused by more unused space in the data blocks.

Just as with PCTFREE, it is worth the effort to look at each table individually and determine a value for PCTUSED that more accurately fits your application. Doing so can improve your system's performance.

A Review of the PCTFREE and PCTUSED Options

As with most tuning efforts, there is a tradeoff between space usage and performance. Here are a few things to keep in mind when you are adjusting the values for PCTFREE and PCTUSED:

- ◆ Because $(100 - \text{PCTFREE})$ and PCTUSED are used as high water and low water marks, the sum of the two cannot exceed 100. On the other hand, the sum does not have to equal 100.
- ◆ The closer the sum of the two values gets to 100, or the closer $(100 - \text{PCTFREE})$ is to PCTUSED, the more overhead is incurred and the more efficient space usage is.
- ◆ If the sum of PCTFREE and PCTUSED equals 100, Oracle attempts to keep exactly PCTFREE free space, increasing CPU usage.
- ◆ Fixed block overhead is not included in the calculation of PCTFREE and PCTUSED.
- ◆ A good compromise of performance and space is to keep approximately one row difference between PCTFREE and PCTUSED. For example, with a 100-byte row in a 2048-byte data block with 100 bytes of overhead, use the following formula:

```
Space = 2048 - 100 = 1948
Average Row = 100 or 5 % of the data block
( 100 - PCTFREE ) - PCTUSED = 5
```

A large difference between $(100 - \text{PCTFREE})$ and PCTUSED means more empty space. Leaving one row difference is usually sufficient.

Here are a few guidelines to follow when adjusting PCTFREE and PCTUSED:

- ◆ **Update Activity, High Row Growth**

If your application frequently uses updates that affect the size of rows, set PCTFREE fairly high and set PCTUSED fairly low. This arrangement allows for a large amount of space in the data blocks for row size growth. For example:

```
PCTFREE = 20-25
PCTUSED = 35-40
( 100 - PCTFREE ) - PCTUSED = 35 to 45
```

◆ **Insert Activity, Small Row Growth**

If most inserts are new rows and there is very little update with row growth, set `PCTFREE` low with set a moderate value for `PCTUSED` to avoid chaining of new rows. This arrangement allows new rows to be inserted into the data block until the point at which more insertions are likely to cause migration or chaining. When this point is reached, no more insertions occur until there is a fair amount of space left in the block; then inserting can resume.

`PCTFREE = 5-10`
`PCTUSED = 50-60`
`(100 - PCTFREE _) - PCTUSED = 30 to 45`

◆ **Performance Primary, Space Abundant**

If performance is critical and you have plenty of space available, you can ensure that migration and chaining never occur by setting `PCTFREE` very high and `PCTUSED` extremely low. Although this can waste quite a bit of space, all chaining and migration should be avoided.

`PCTFREE = 30`
`PCTUSED = 30`
`(100 - PCTFREE _) - PCTUSED = 40`

◆ **Space Critical, Performance Secondary**

If you are have very large tables or if you have moderate-sized tables and disk space is at a premium, set `PCTFREE` very low and `PCTUSED` very high. This arrangement ensures that you take maximum advantage of the available space. Note that you will see some performance loss caused by increased chaining and migration.

`PCTFREE = 5`
`PCTUSED = 90`
`(100 - PCTFREE _) - PCTUSED = 5`

A Review of I/O Reduction Techniques

Reduction of I/O is important for performance because I/O is one of the slowest operations in the computer system. Caching data is many times faster than an access from disk. In fact, a read from memory can take place over 100,000 times faster than a read from disk.

You have seen that you can speed up I/O by separating the sequential and the random I/Os and separating data from indexes to get maximum concurrency. You have also seen how to speed up I/Os by avoiding them altogether—such as by reducing dynamic extensions and by avoiding migrated and chained rows.

As you see in later chapters, the speed of I/O from disk is a fixed value that you can work around only by caching, reducing contention, and avoiding I/Os if possible.

Tuning Rollback Segments

Another area of contention may be in the rollback segments. Rollback segments are constantly used during transaction processing; any delays caused by contention on rollback segments affect performance.

Rollback segments record transactional information that may be used in the event that the transaction should be rolled back. Rollback segments are also used to provide read consistency and are used for database recovery.

Read consistency allows a long-running transaction to always obtain the same data within the query. During the transaction, the data is consistent to a single point in time and does not change. Even though the data may have changed, and perhaps the DBWR may even have written it out, other transactions do not see those changes until a `COMMIT` has occurred. In fact, only transactions that start after this transaction has been committed see those changes.

Rollback segments must be carefully watched and can be tuned in several ways. It is important not only to size the rollback segments correctly but also to create the proper number of rollback segments and properly distribute them according to the number of user processes that require them. The next section discusses how the rollback segments operate.

Understanding How Rollback Segments Work

As a transaction is being processed, information relating to changes made to the data files by that transaction is constantly being written to the rollback segments. It is important that this information be saved because a rollback would require that all data be restored to its original condition.

The information written by the transaction to the rollback segments is called *rollback entries*. Depending on the length of the transaction and the amount of changes to data, there may be more than one rollback entry for each transaction. These entries are linked together so that they can easily be used in the event of a rollback.

The information stored in the rollback segments includes block information about what blocks have been modified and the data as it was before the change occurred. Remember that the redo log also records information about changes in the database. The redo log along with the rollback segments can restore your data up to the point of failure.

Rollback segments are concurrently used by one or more transactions. You can tune the rollback segments to provide for optimal efficiency and space usage. More transactions sharing rollback segments cause more contention and use space more efficiently. Fewer transactions per rollback segment cause less contention and waste more space.

Oracle maintains what is called a *transaction table* for each rollback segment. The transaction table stores information about what transactions use that rollback segment and the rollback entries for each change done by these transactions.

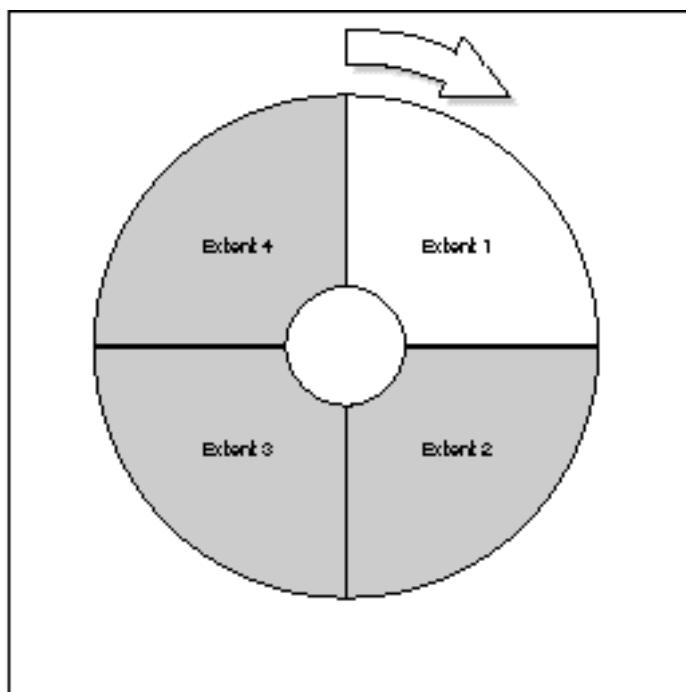
Each time a new transaction begins, it is assigned to a rollback segment. This can happen in one of two ways:

- ◆ **Automatic.** Oracle automatically assigns the transaction a rollback segment. The assignment takes place when the first DDL or DML statement is issued. Queries are never assigned rollback segments.
- ◆ **Manual.** The application can manually specify a rollback segment by using the `SET TRANSACTION` command with the `USE ROLLBACK SEGMENT` parameter. This arrangement allows the developer to choose the correct size rollback segment for a particular task. The rollback segment is assigned for the duration of the transaction.

At the end of each transaction, when the `COMMIT` has occurred, the rollback information is released from the rollback segment—but is not deleted, in order to provide read-consistent views for other queries that started before the transaction was committed. To retain this information as long as possible, the rollback segments are written as a circular buffer as described next.

You can think of rollback segments as a sort of *circular buffer*. A rollback segment must have at least two extents (usually more). When a transaction fills up one extent, it starts using the next extent in sequence. When it gets to the last extent, the transaction continues with extent 1 again, if it is available (see Figure 9.3).

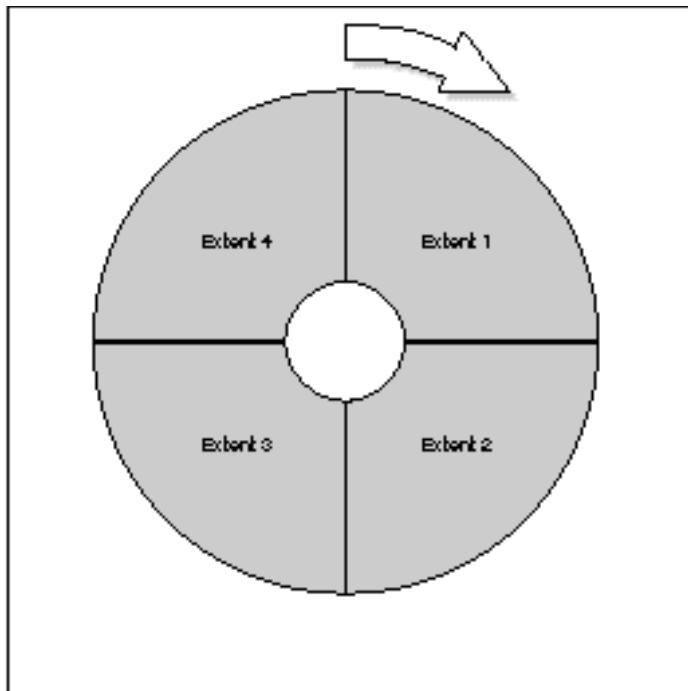
Figure 9.3
A rollback segment.



If the transaction uses the last extent in the segment, it looks to see whether the first extent is available. If not, another extent is created (see Figures 9.4 and 9.5).

Figure 9.4

A rollback segment with all extents used.



The number of extents used for the rollback segments is determined in the definition of the rollback segments when you create them. Rollback segments are created with the following command:

```
CREATE PUBLIC ROLLBACK SEGMENT rsname
TABLESPACE tname
STORAGE (
  INITIAL 100K
  NEXT 100K
  OPTIMAL 1500K
  MINEXTENTS 15
  MAXEXTENTS 100);
```

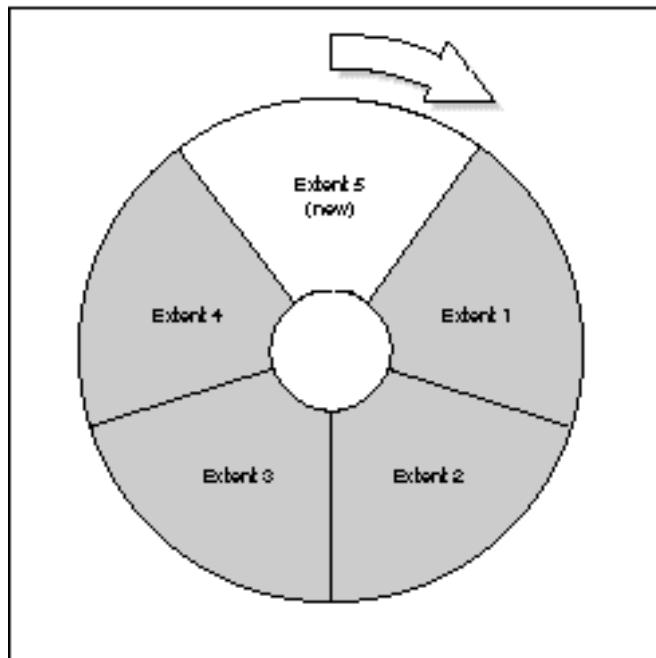
In this example, the following values are specified by the user:

- ◆ **INITIAL:** The initial extent is 100K.
- ◆ **NEXT:** The second extent is 100K. With rollback segments, it is always a good idea to make all extents the same size because there is no distinction between different extents.
- ◆ **OPTIMAL:** Set to 800K. This value specifies what you think the optimal size for the rollback segment should be. When extents are no longer needed, they are eliminated until this size is reached.

- ◆ **MINEXTENTS:** Set to 8. The minimum number of extents. This is also the number allocated when the segment is created.
- ◆ **MAXEXTENTS:** Set to 100. The maximum number of extents that can be dynamically allocated.

Figure 9.5

A rollback segment showing dynamic growth.



Initially, there are **MINEXTENTS** number of extents in the rollback segment. As extents fill up, they are used in a circular fashion, returning to the first extent when all others are filled. If a rollback segment has used all the space in all the extents and **MAXEXTENTS** has not been reached, another extent is created. If the size of the rollback segment is larger than **OPTIMAL** and there are unused extents, the unused extents are dropped from the rollback segment.

Both the creation and destruction of rollback segment extents cause overhead in the system. In addition to the overhead created by the addition of extents to a rollback segment, the transaction needing to write into that rollback segment must wait for the extent to be created before it can continue. The following sections explain how to tune your use of rollback segments.

Tuning Rollback Segments

To properly configure a system's rollback segments, you must create enough rollback segments and they must be of a sufficient size.

Number of Rollback Segments

The number of rollback segments should be determined by the number of concurrent transactions in the database. Remember, the fewer transactions per rollback segment, the less contention. A good rule of thumb is to create about one rollback segment for every four concurrent transactions.

Rollback contention occurs when too many transactions try to use the same rollback segment at the same time and some of them have to wait. You can tell whether you are seeing contention on rollback segments by looking at the dynamic performance table, V\$WAITSTAT. V\$WAITSTAT contains the following data related to rollback segments:

- ◆ *UNDO HEADER*: The number of waits for buffers containing rollback header blocks.
- ◆ *UNDO BLOCK*: The number of waits for buffers containing rollback blocks other than header blocks.
- ◆ *SYSTEM UNDO HEADER*: Same as UNDO HEADER for the SYSTEM rollback segment.
- ◆ *SYSTEM UNDO BLOCK*: Same as UNDO BLOCK for the SYSTEM rollback segment.

The *system rollback segment* is the original rollback segment created when the database was created. This rollback segment is used primarily for special system functions but is sometimes used when no other rollback segment is available. Typically, the system rollback segment is not used and you do not have to be concerned about it.

You can view these values with the following SQL statement:

```
SQL> SELECT class, count
  2  FROM V$WAITSTAT
  3  WHERE class IN
  4  ('undo header', 'undo block', 'system undo header', 'system undo block');

CLASS          COUNT
-----
system undo header      0
system undo block       0
undo header            0
undo block             0
```

Compare these values with the total number of requests for data. Remember (from earlier in the chapter) that the number of requests for data is equal to the sum of DB BUFFER GETS and CONSISTENT GETS from V\$SYSSTAT. Also remember that you can extract that information with the following query:

```
SQL> SELECT SUM(value) "Data Requests"
  2  FROM v$sysstat
  3  WHERE name IN ('db block gets', 'consistent gets');

Data Requests
-----
5105
```

If the number of waits for any of the rollback segment blocks or headers exceeds more than 1 percent of the total number of requests, you should reduce the contention by adding more rollback segments.

Size of Rollback Segments

Create several different sizes of rollback segments. Each type of rollback segment should be used by the application developer based on the type and length of the transaction:

<i>Transaction Type</i>	<i>Comments</i>
OLTP	<p>OLTP transactions are characterized by many concurrent transactions, each modifying perhaps only a small amount of data. These types of transactions benefit from a reduction of contention and quick access from cached rollback segments. Try to create many small rollback segments of perhaps 10K to 20K in size, each with 2 to 4 extents—optimally with a rollback segment available for each transaction.</p> <p>The small size of the rollback segments provides for a better chance of being cached in the SGA. There is probably very little dynamic growth of the extents.</p>
Long Queries	<p>For long queries where read consistency calls for quite a bit of rollback information to be accessed, use a larger rollback segment. A good rule of thumb is to create rollback segments approximately 10 percent the size of the largest table (most SQL statements affect only about 10 percent of the data in a table).</p>
Large Updates	<p>For transactions that update large amounts of data, you should also use a larger rollback segment. As is the case with the long queries, it is appropriate to create rollback segments approximately 10 percent the size of the largest table.</p>

Size and Number of Extents

In general, the best rollback I/O performance can be obtained when there are approximately 10 to 20 extents of equal size per rollback segment. To determine the size and number of extents, use the following formula:

```
Rollback Segment Size = Rsize = Size of largest table / 10  
Number of Extents = NE = 10  
Size of Extents = Esize = Rsize / NE
```

When creating the rollback segments, use the value of `Esize` for `INITIAL` and `NEXT`; use the value of `NE` for `MINEXTENTS`. Even when using these rules, you may not achieve the most effective size for your rollback segments. If dynamic growth is occurring, you may be losing performance.

Avoid Dynamic Growth

As stated earlier, you want to avoid the dynamic space management that causes additional overhead and transactional delays. To determine whether rollback segments are a problem, look in the dynamic performance table, `V$ROLLSTAT`. The following columns are of particular interest:

- ◆ `EXTENTS`: Number of rollback extents.
- ◆ `RSSIZE`: The size in bytes of the rollback segment.
- ◆ `OPTSIZE`: The size to which `OPTIMAL` was set.
- ◆ `AVEACTIVE`: The current average size of active extents. *Active extents* are defined as extents with uncommitted transaction data.
- ◆ `AVESHINK`: The total size of free extents divided by the number of *shrinks* (see the second item following).
- ◆ `EXTENDS`: The number of times the rollback segment added an extent.
- ◆ `SHRINKS`: The number of times the rollback segment shrank. Each shrink may be one or more extents at a time.
- ◆ `HWMSIZE`: The high water mark of rollback segment size. This is the largest that the segment size ever grew to be.

You can look at these statistics by using a SQL statement such as this one:

```
SQL> SELECT substr(name,1,40), extents, rssize, aveactive, aveshrink, extends,
  shrinks
  2  FROM v$rollname rn, v$rollstat rs
  3  WHERE rn.usn = rs.usn;
```

SUBSTR(NAME,1,40)	EXTENTS	RSSIZE	AVEACTIVE	AVESHINK	EXTENDS
SHRINKS	-----	-----	-----	-----	-----
SYSTEM 0	4	202752	0	0	0
RB_TEMP 0	53	540672	23929	0	0
RB1 0	2	202752	0	0	0
RB2 0	2	202752	55193	0	0

If the average size is close to the size set for `OPTIMAL`, `OPTIMAL` is set correctly. If either extends or shrinks are high, you must increase the value for `OPTIMAL`.

Review of Rollback Segment Tuning

To optimize rollback segments, remember the following rules:

- ◆ Avoid using the default rollback segments in the `SYSTEM` tablespace. Create your own tablespace for rollback segments.
- ◆ Create approximately one rollback segment for every four concurrent transactions you expect to have.
- ◆ Create 10 to 20 extents for each rollback segment (except for short-running OLTP transactions).
- ◆ Assign large rollback segments to long-running queries. These queries may need a large rollback to reconstruct a read-consistent version of the data. This data must be available for the entire length of the transaction. The size of the rollback segments should be approximately 10 percent the size of the largest table.
- ◆ Assign large rollback segments for long transactions that modify a large amount of data. These transactions create large amounts of rollback information. The size of the rollback segments should be approximately 10 percent the size of the largest table.
- ◆ Avoid dynamic expansion and reduction of rollback space by monitoring rollback information.

Rollback segments should be monitored periodically because the applications and user activity changes. What is optimal for rollback segments today may degrade over time because of an increase in user activity or changes in application code.

Checking for Latch Contention

For a transaction to be completed, redo information must be written to the redo log. Until this write has occurred, there is danger of losing the transaction should some sort of instance failure occur. To avoid this condition, a `COMMIT` is not completed until the redo record has been written.

Any kind of bottleneck in the redo log can cause performance problems for every process on the system. To make sure that this does not happen, check for contention on the redo log buffer latches as well as contention on the redo log buffers.

Redo Log Buffer Contention

To check for contention on the redo log buffer, simply check the dynamic performance table, `V$SYSSTAT`, for redo log space requests. If this number is not zero, this indicates that a process had to wait for space in the redo log buffer; the size of the redo log buffer should be increased. Check for this condition with the following SQL statement:

```
SQL> SELECT name, value
  2  FROM v$sysstat
  3 WHERE name = 'redo log space requests';
```

NAME	VALUE
redo log space requests	0

If this value is not 0, increase the initialization parameter `LOG_BUFFER` by 5 to 10 percent until your system runs with redo log space requests close to 0.

Redo Log Buffer Latch Contention

The access to the redo log buffer is controlled by two types of latches: the redo allocation latch and redo copy latches.

The *redo allocation latch* controls the writing of redo entries to the redo log buffer. To write an entry into the redo log buffer, the user process must obtain the redo allocation latch, allocate space in the redo log buffer, and then copy the entry into the buffer. Whenever the user process has finished copying into the redo log buffer, it releases the latch, thus allowing other user processes to use the redo allocation latch.

Because only one redo allocation latch exists, only one user process can write to the log buffer at a time. The maximum amount of data that can be written into the latch is specified by the initialization parameter `LOG_SMALL_ENTRY_MAX_SIZE`.

If the redo entry exceeds the value specified by `LOG_SMALL_ENTRY_SIZE`, the user process must obtain a *redo copy latch* before copying into the redo log buffer. After allocating a redo copy latch, the user process can write into the buffer.

With multiple-CPU machines, you can have multiple redo copy latches. That is, you can allow simultaneous entries into the redo log buffer, which increases performance. The number of redo copy latches is determined at instance startup by the parameter `LOG_SIMULTANEOUS_COPIES`; the value defaults to `CPU_COUNT`. In a single-CPU machine, there are no redo copy latches and all access to the redo log buffer is through the redo allocation latch, regardless of size.

There are two ways of accessing both the redo allocation latch and the redo copy latches:

- ◆ **Willing-to-wait.** The process requests a latch; if it is not available, the process sleeps for a while and tries again later. The process continues waiting until it gets the latch.
- ◆ **Immediate.** The process either gets the latch or continues processing if it cannot get it.

Determining Latch Contention Problems

Latch contention can be determined by examining the dynamic performance table, `V$LATCH`. The significant values are given here:

- ◆ *GETS*: For willing-to-wait requests. The number of successful requests.
- ◆ *MISSES*: For willing-to-wait requests. The number of times the initial request fails.
- ◆ *SLEEPS*: For willing-to-wait requests. The number of times subsequent requests fail.
- ◆ *IMMEDIATE_GETS*: For immediate requests. The number of successful requests.
- ◆ *IMMEDIATE_MISSES*: For immediate requests. The number of times the request fails.

You can look for contention on latches by using the following SQL statement:

```
SQL> SELECT SUBSTR(name,1,20), gets, misses, sleeps, immediate_gets,
  immediate_misses
  2  FROM v$Latch
  3 WHERE name IN ('redo allocation', 'redo copy');

SUBSTR(NAME,1,20)      GETS      MISSES      SLEEPS IMMEDIATE_GETS IMMEDIATE_MISSES
-----  -----  -----  -----  -----  -----
redo allocation        2744          0          0          0          0
redo copy                  0          0          0          0          0
```

If the ratio of *MISSES* to *GETS* exceeds 1 percent or the ratio of *IMMEDIATE_MISSES* to the sum of *IMMEDIATE_MISSES* and *IMMEDIATE_GETS* exceeds 1 percent, contention is probably affecting performance in your system.

Solving Latch Contention Problems

If latch contention is causing performance problems, there are several things you can do to try to reduce this contention.

To reduce contention on the redo allocation latch, you can reduce the size of *LOG_SMALL_ENTRY_MAX_SIZE* to minimize the time any user process holds the latch. This is done by reducing both the size and number of redo entries copied onto the redo allocation latch.

Resetting the value of *LOG_SIMULTANEOUS_COPIES* can reduce contention by adding more redo copy latches. You may see improvement by adding up to twice the number of redo copy latches as CPUs.

Another way to reduce the time a user process holds the latch is by requiring the user process to prebuild the redo entries before it obtains the latch. This prebuilt entry reduces contention on the latch by not using CPU time to build the entry while the latch is being held.

By setting the initialization parameter *LOG_ENTRY_PREBUILD_THRESHOLD*, any redo entry of a smaller size than this parameter must be prebuilt. The default for *LOG_ENTRY_PREBUILD_THRESHOLD* is 0 (that is, it requires no prebuilding).

Although redo log buffer latches are usually not a problem, they can be a source of contention under certain circumstances. You have seen how to check for latch contention and how to reduce contention if you find it. Unless you are under extremely heavy loads, it is not necessary to monitor the redo buffer latches.

Tuning Checkpoints

Checkpoints occur in the Oracle database to ensure that all “dirty blocks” in the SGA are eventually written to disk. Because the DBWR works on a least recently used (LRU) algorithm, data that has been untouched in the SGA for the longest time is written out first. This has the side effect that often-used data blocks may never be written out to disk.

A checkpoint causes all the dirty blocks in the SGA to be written to disk. The following situations can cause a checkpoint:

- ◆ Every log switch causes a checkpoint. If a checkpoint is in progress, the log switch overrides it and the checkpoint will restart.
- ◆ The initialization parameter `LOG_CHECKPOINT_INTERVAL` can be set to force a checkpoint when a certain number of redo log blocks have been written relative to the last checkpoint. This is useful when you are running with large log files and want several checkpoints between log switches.
- ◆ The initialization parameter `LOG_CHECKPOINT_TIMEOUT` can be set to cause a checkpoint to occur a specific number of seconds since the beginning of the last checkpoint. This parameter can be used for large redo log files that have a desired checkpoint frequency.
- ◆ A checkpoint is forced at the beginning of a tablespace backup only on the affected data files. This checkpoint also overrides any other checkpoint that is in progress.
- ◆ A checkpoint is forced on the data files of a tablespace that is brought offline.
- ◆ A checkpoint is forced on an instance shutdown. If the administrator shuts down the instance with `SHUTDOWN NORMAL` or `SHUTDOWN IMMEDIATE`, a checkpoint occurs.
- ◆ A checkpoint can be caused manually by the administrator. This checkpoint overrides any in-progress checkpoints.

Depending on how a checkpoint is invoked, a checkpoint can be either a “normal” or “fast” checkpoint. For a normal checkpoint, a few more blocks of data are written by the DBWR when it writes, on behalf of the checkpoint. For a fast checkpoint, a large number of blocks are written by the DBWR when it writes, on behalf of the checkpoint. These blocks are tagged on to the normal activity of the DBWR; while checkpoint blocks are being written, normal DBWR activity is still being done.

In the case of the normal checkpoint, transaction processing is not as affected as it is with the fast checkpoint. With the normal checkpoint, more I/Os are required to complete the checkpoint but the throughput of the system is not drastically affected. Normal checkpoints are invoked by log switches and checkpoint intervals set up with the initialization parameters.

If you notice a drop in performance as a normal checkpoint completes, this can sometimes be compensated for by enabling the CKPT process. Doing so can help if the performance drop is caused by the LGWR process being too busy updating file headers. CKPT updates the file headers, leaving LGWR free to write out the redo entries. To enable the CKPT process, set the initialization parameter `CHECKPOINT_PROCESS` to TRUE.

With fast checkpoints, transaction processing is affected more drastically. Because a large number of blocks are added to the DBWR's normal activity, the number of I/Os are reduced but the DBWR is affected by this process. Because the DBWR is busy with the checkpoint, its normal activities are usually affected, causing increased response times and decreased throughput. Fast checkpoints are caused by online tablespace backups, instance shutdowns, and administrator-forced checkpoints.

If you are not as interested in recovery time as you are in throughput, you can set the checkpoint frequency so that a checkpoint is performed only when the redo log switch occurs. If you are interested in quick recovery, set the checkpoint frequency more often. To further increase the checkpoint interval, increase the size of the redo log files.

Optimizing Archiving

Archiving is a normal part of daily operations and should be tuned to have the least effect on the performance of the system. Archiving is done when the instance is running in ARCHIVELOG mode. I recommend that you always run in ARCHIVELOG mode. The benefits of running in this mode are as follows:

- ◆ Allows recovery of transactions from the last backup until the instance failure.
- ◆ Allows for online backups. Running in ARCHIVELOG mode allows you to perform backups while users are still accessing the database.

You can archive manually or you can set the system up to archive automatically. Automatic archiving is enabled by setting the initialization parameter `LOG_ARCHIVE_START` to TRUE. Typically, archiving is done automatically with the archive process starting as soon as a log switch has occurred.

Archiving usually has very little effect on the performance of the system—but there *are* some cases in which performance can be affected. In very large databases, the archive can have an effect on system performance. Physical placement of the log files can also cause performance problems.

Remember that once the log switch has occurred, the archive process begins to sequentially read from the previous log file and write that information sequentially to the archive log destination. If multiple log files are on the same disk drive, you may actually cause a large amount of random disk activity switching between the current log file and the log file being archived.

TIP: If you place log files on different disk drives, one drive can be logging sequentially while the previous log drive is being archived sequentially. Accessing the disk drives sequentially greatly enhances performance.

Adjusting the Effect of Archiving

By adjusting the initialization parameters `LOG_ARCHIVE_BUFFERS` and `LOG_ARCHIVE_BUFFER_SIZE`, you can either slow down or speed up the performance of archiving. By speeding up archiving, the effect on the system is of a shorter duration but is more noticeable. Slowing down archiving lengthens the duration but reduces the effect.

To speed up archiving, use multiple archive buffers by setting `LOG_ARCHIVE_BUFFERS` to 2 or more and slowly increase the value of `LOG_ARCHIVE_BUFFER_SIZE` until you get the performance you want. Multiple buffers guarantee that a device such as a tape backup unit always has data available (that is, the device is streamed).

To slow down archiving, set the value of `LOG_ARCHIVE_BUFFERS` to 1. Set the value of `LOG_ARCHIVE_BUFFER_SIZE` to the maximum allowed by your OS. If archiving is still happening too quickly, reduce the size of `LOG_ARCHIVE_BUFFER_SIZE` until you get the performance you desire.

CAUTION: If archiving is configured to run too slowly, you may run into a situation in which a redo log is next in line for reuse but has not yet finished archiving. If this occurs, transaction processing stops until the archive finishes. You can avoid this situation by adding more redo log files to the redo log file group.

Archiving is a necessary part of day-to-day operations that usually does not affect the system. For the cases in which archiving does adversely affect performance, slight configuration changes can usually solve the problem.

Optimizing Sorts

The default sort area size is usually sufficient for most applications. If your application often does sorts that do not fit into memory, you may want to adjust the size of the sort area. One way to determine whether your sorts are occurring in memory or on disk is to look at the dynamic performance table, `V$SYSSTAT`. The statistics of interest are given here:

- ◆ *Sorts (memory)*: The number of sorts that were able to fit in the in-memory sort area.
- ◆ *Sorts (disk)*: The number of sorts that required temporary space on disk.

You can look at these parameters by using the following SQL statement:

```
SQL> SELECT name, value
  2  FROM v$sysstat
  3 WHERE name IN ('sorts (memory)', 'sorts (disk)');
```

NAME	VALUE
sorts (memory)	22
sorts (disk)	0

If you have a lot of sorts that use temporary space on disk, you may see a performance improvement by increasing the size of the sort area in memory. Do this by increasing the Oracle initialization parameter, `SORT_AREA_SIZE`. By increasing the sort area size, you see a benefit from faster sorts and less disk I/O.

CAUTION: Sorts occur in user memory, not in the SGA. Be careful that multiple simultaneous sorts do not cause paging and swapping in your system. Be sure that there is enough free memory available for your sorting activities.

If you increase the value for `SORT_AREA_SIZE`, you may want to decrease the value of `SORT_AREA_RETAINED_SIZE` (the amount of session memory retained for sorts). When Oracle performs sorts, it uses the amount of memory defined by `SORT_AREA_SIZE`. When the sort is completed, Oracle deallocates the memory, leaving as much as it can of the sort output in memory. Oracle deallocates memory until it reaches the value defined by `SORT_AREA_RETAINED_SIZE`.

If the sort cannot fit into `SORT_AREA_SIZE`, the sort is broken into smaller pieces, which are sorted individually. These individual sorted pieces are known as *runs*.

Usually, sorts are not a problem. However, you should periodically check the statistics and take appropriate action if necessary. Be sure to periodically monitor system memory usage as well.

Minimizing Free List Contention

One other area of possible contention is in the free list. The *free list* is maintained to provide a faster mechanism to get free data blocks from the buffer cache. The free list contains a linked list of blocks in the segment that has space available.

Contention on the free list can be determined by looking at the dynamic performance table, V\$WAITSTAT. Use the following SQL statement to obtain this information:

```
SQL> SELECT class, count
  2  FROM v$waitstat
  3 WHERE class = 'free list'
```

CLASS	COUNT
-----	-----
free list	0

If the free list number is greater than 1 percent of the total number of requests, you should add more free lists. Remember that the total number of requests is determined from the sum of database block gets and consistent gets from V\$SYSSTAT as shown here:

```
SQL> SELECT SUM(value) "Data Requests"
  2  FROM v$sysstat
  3 WHERE name IN ('db block gets', 'consistent gets');
```

Data Requests

5105

You can add more free lists by re-creating the table with a larger value for the `FREELISTS` storage parameter. You may want to increase this parameter to the number of concurrent `INSERT` transactions that you expect to see.

Although there usually is not a problem with free lists, monitoring them periodically will alert you if the system load has increased to where free lists are a problem.

Summary

This chapter looked at several areas you can improve in the Oracle instance. Most of these areas have a general theme; this theme relates to the fact that disk I/O is many times slower than memory.

If you can optimize an instance to take advantage of this fact, you can achieve optimal performance. Here is a summary of the goals discussed in this chapter:

- ◆ **Tune for cache hits.** By sizing `DB_BLOCK_BUFFERS` and `SHARED_POOL_SIZE` to get good cache-hit rates, I/Os are reduced.
- ◆ **Separate sequential and random I/O.** Doing so speeds up access to disk and optimizes the disk subsystem.
- ◆ **Avoid unnecessary I/Os.** Reduce dynamic space allocation. Also reduce chained and migrated rows.
- ◆ **Minimize contention.** This is true for rollback segments, free lists, and redo buffer latches.
- ◆ **Sort in memory.** If you have memory available, allow sorts to occur in memory, thus increasing sort performance and reducing I/Os.
- ◆ **Tune archiving.** You should always archive. Tune the archive process to have the least effect on your system.
- ◆ **Tune checkpoints.** Adjust the checkpoint interval based on your needs (remember that there is a performance-versus-recovery interval issue here).

These areas of instance tuning can all help the overall performance of the system. The next chapter looks at some additional areas where you can change the system to increase performance.

Chapter

10

Performance Enhancements

This chapter continues where the last chapter left off. This chapter continues to look at the Oracle server and additional things you can do to improve system performance. Because these items are essentially an unrelated list of individual topics, the chapter is arranged alphabetically. Don't interpret the order of the topics to mean that one of these is more important than another.

First, the chapter looks at why you may want to change the block size of the database. Then it looks in detail at the concept of clusters and how they can benefit your configuration. The chapter also looks at indexes: when you should use one and how.

You also look at some of the optional products Oracle has to offer (such as the Parallel Query option and the Parallel Server option) and how they are used.

At the end of the chapter are several miscellaneous items such as spin count that may affect performance in your system.

Block Size

Typically, the default block size is sufficient for all applications. In a few cases, however, increasing the size of the database buffers is beneficial. Depending on your operating system, there are different limitations on the value of `DB_BLOCK_SIZE`.

If you have a large system with many disks, think about increasing the database block size. These very large systems typically use a block size of 4K or larger. The larger block size may benefit the overall performance of the system.

The size of the database block has several effects of which you should be aware. If the block size is too small for the data stored in the database, you suffer the following effects:

- ◆ Unnecessary I/O, caused by the additional chaining and migration of rows that do not fit in the block.
- ◆ Unnecessary latch operations, caused by redo information not fitting in the redo log buffer.
- ◆ Wasted space, caused because the smaller blocks are less likely to be entirely filled.

All these problems result in unnecessary overhead in space, CPU, and I/Os. By increasing the block size, many of these problems can be eliminated.

But be careful. A block size that is too large can cause the following effects on your system:

- ◆ **More I/Os.** Because the block size is larger, there are fewer database block buffers in memory for the same size SGA. This situation results in more I/Os because buffers must be written out to make room for new ones.
- ◆ **Larger I/Os.** The larger the I/O, the longer it takes to copy data into memory. But if the large amount of data is needed, one long I/O is much better than two shorter I/Os. See Chapter 14, “Advanced Disk I/O Concepts,” for more information on disk retrieval time.
- ◆ **Disk queuing takes longer.** Because of the longer I/Os, disk queuing takes longer, which is also worthwhile if the data you retrieved is useful.

Other effects of a larger block size include a reduction in the depth of the B*-tree index and less disk-seek overhead to retrieve data blocks. Another effect of a larger block size is an increase in the size of cluster buckets.

Here are a few guidelines to help you decide whether changing the size of `DB_BLOCK_SIZE` can benefit you:

- ◆ **OLTP systems benefit from smaller blocks.** If your application is OLTP in nature, you will not benefit from larger blocks. OLTP data typically fits well in the default block size; larger blocks unnecessarily eject blocks from the SGA.
- ◆ **DSS systems benefit from larger blocks.** In the DSS system where table scans are common, retrieving more data at a time results in a performance increase.
- ◆ **Larger databases benefit from larger blocks.** Larger databases see a space benefit because there is less waste per block.
- ◆ **Databases with large rows benefit from larger blocks.** If your rows are extremely large (such as images or text) and don't fit in the default block, you will see a definite benefit from a larger block size.

When changing the block size, be careful to set `DB_BLOCK_SIZE` only to a multiple of the OS block size. Doing so guarantees that you are not reading OS blocks and only using part of the block. For example, if you use 1K Oracle blocks and 2K OS blocks, the OS reads 2K from the disk every time you request a 1K block.

The database block size should be changed only after careful consideration. If you see excessive chained rows or wasted space, consider a larger block size. If you are running a DSS system, you may also want to consider a larger block size. Larger block sizes can be very beneficial—but can be detrimental if you change them unnecessarily.

Clusters

A *cluster*, sometimes called an *index cluster*, is an optional method of storing tables in an Oracle database. Within a cluster, multiple related tables are stored together to improve access time to the related items. Clusters are really useful only when the related data is often accessed together. The existence of a cluster is transparent to users and to applications; the cluster affects only how data is stored.

The use of a cluster can be advantageous in certain situations and disadvantageous in others. Be careful when determining whether a cluster can help performance in your configuration. Typically, clusters are advantageous if the related data that is clustered is primarily used in joins.

If you have two tables with related data that are frequently accessed together, using clusters can improve performance by preloading the related data into the SGA. Because you frequently use the data together, having that data already in the SGA greatly reduces access time.

Clusters are beneficial when joins occur on the cluster data because all the data is retrieved in one I/O operation. Following is an example of where a cluster would be beneficial.

Suppose that you are keeping a database of information for a dog club. (Because I am a dog lover and have several dogs of my own, I can easily relate to this topic.) In this database, you want to keep track of all the dogs and their owners as well as some information about each of the dogs. To do this, you must create several tables.

First, you need a table of all the dogs who are members of the dog club. You also need a table of the dog breeds and some information about the dog breeds (see Figure 10.1).

Figure 10.1

The DOGS table and the BREEDS table.

DOGS Table

DOGNAME	BREED	OWNER
Dash	1	Jones
Chip	1	Jones
Ruby	3	Smith
Duncan	2	Miller
Rufus	3	Smith
Splash	4	Blake
Piper	1	Turner
Pierce	1	King
Sheba	5	Adams
Toller	1	King
Spots	2	Ward
B.J.	4	Allan

BREEDS Table

BREED	DESCRIPTION
1	Border Collie
2	Sheltie
3	Jack Russell Terrier
4	Golden Retriever
5	All American (Mix)

By combining the two tables into a cluster, you can save time when retrieving the data (because the breed information for a particular dog is essentially read into the SGA when the information for that dog is). The common columns of the cluster are called the *cluster key*, and must be indexed.

Figure 10.2 shows what the table looks like as a cluster. Note that the cluster key is the breed identification number.

If this information is frequently used together, this cluster arrangement is a performance win. The cluster has the ease of use of individual tables but the additional performance of the cluster.

If you do not typically use information together, there is no performance benefit of using a cluster. There is even a slight disadvantage because additional SGA space is taken up by the additional table information.

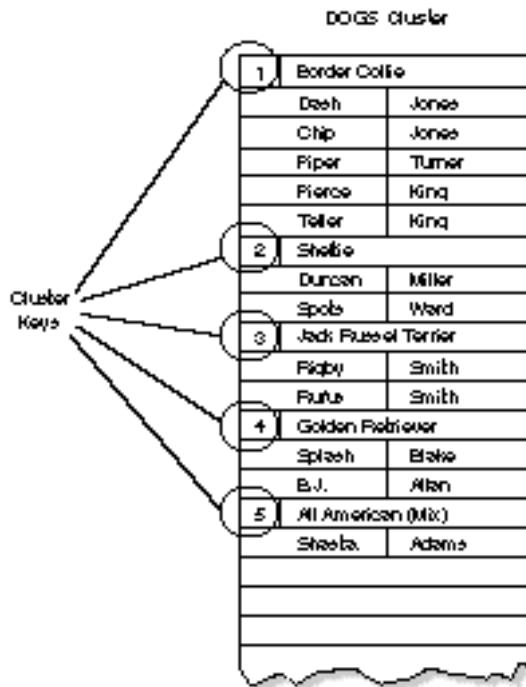
An additional disadvantage of clusters is a reduction of the performance of `INSERT` statements. This performance loss is caused by the additional complexity of the use of space and the fact that there are multiple tables in the same block. The clustered table also spans more blocks than the individual tables, thus causing more data to be scanned.

In summary, a cluster can be useful for tables where data is primarily accessed together in a join. The reduced I/O needed to bring the additional data into the SGA and the fact that the data is already cached can be a big advantage.

If the tables have a large number of `INSERT` statements or if the data is not frequently accessed together, a cluster is not useful and should not be used.

Figure 10.2

The DOGS and BREEDS tables as a cluster.



Do not cluster tables if full-table scans are often performed on only one of the tables in the cluster. The additional space required by the cluster and the additional I/O reduce performance.

Direct-Write Sorts

Using direct writes causes the server processes to write the output of sort operations directly to disk, bypassing the buffer cache. The effect of direct writes is that, for sort operations, large amounts of block buffers are not ejected from the buffer cache. This leaves the buffer cache available for normal queries and updates. By setting the Oracle initialization parameter `SORT_DIRECT_WRITES` to TRUE, you enable this feature.

Direct-write sorts take more memory than normal sorts. The amount of memory it uses can be determined with the following formula:

$$\text{Direct Write Sort Memory} = (\text{SORT_WRITE_BUFFERS}) * (\text{SORT_WRITE_BUFFER_SIZE})$$

Using direct-write sorts can improve performance both in the sort and in the entire system by not consuming the buffer cache. Only use direct-write sorts if you have sufficient memory and temporary disk space. The temporary disk space should have a sufficient I/O bandwidth to handle the load.

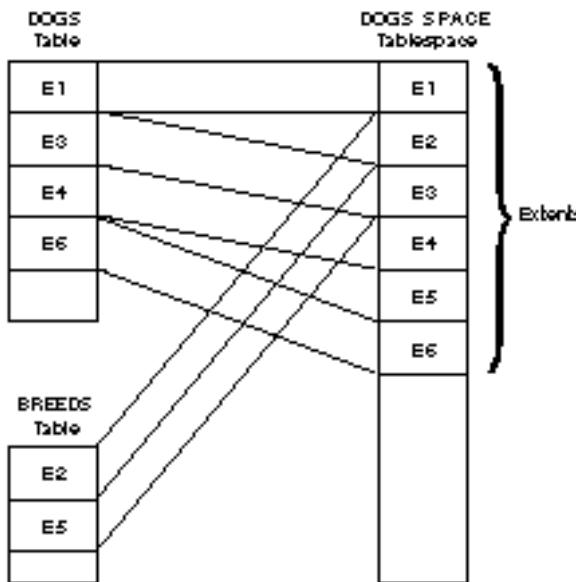
Fragmentation

Fragmentation occurs when pieces of the database are no longer contiguous. Fragmentation can consist of disk fragmentation or tablespace fragmentation. Both types of fragmentation usually affect performance.

Disk fragmentation usually causes multiple I/Os to occur where one I/O should have been sufficient (such as with chained or migrated rows). Disk fragmentation can also be caused when the extents that make up the database segments are noncontiguous, as happens when there is excessive dynamic growth (see Figure 10.3).

Figure 10.3

Disk fragmentation.

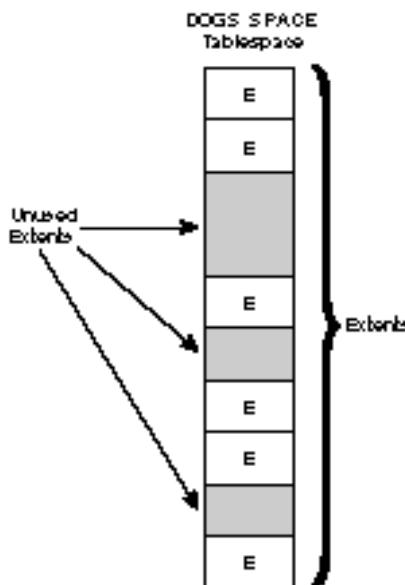


Tablespace fragmentation is caused by the dropping and creation of segments, which can cause large free areas between segments. The free areas result in the inefficient use of space and cause excessive disk seeks over the empty areas (see Figure 10.4). Tablespace fragmentation can also prevent Oracle from taking advantage of multiblock reads.

Tablespace fragmentation can be detected by looking in the Oracle table DBA_EXTENTS with a query such as this one:

```
SQL> SELECT SUBSTR(tablespace_name,1,25) "Tablespace Name",
  2    block_id "Block",
  3    blocks "Number of Blocks",
  4    SUBSTR(segment_name,1,25) "Segment Name"
  5   FROM dba_extents
  6  WHERE tablespace_name = 'SYSTEM'
  7 ORDER BY block_id;
```

Tablespace Name	Block Number	Number of Blocks	Segment Name
SYSTEM	2	25	SYSTEM
SYSTEM	27	25	SYSTEM
SYSTEM	52	60	C_OBJ#
SYSTEM	112	5	I_OBJ#
SYSTEM	117	5	C_TS#
SYSTEM	122	5	I_TS#
SYSTEM	127	10	C_FILE#_BLOCK#
SYSTEM	137	5	I_FILE#_BLOCK#
SYSTEM	142	5	C_USER#
SYSTEM	147	5	I_USER#
SYSTEM	152	5	UNDO\$
SYSTEM	157	5	FILE\$
SYSTEM	162	5	OBJ\$
SYSTEM	167	5	CONS\$
SYSTEM	172	25	C_COBJ#
SYSTEM	197	5	I_COBJ#
SYSTEM	202	5	I_TAB1
SYSTEM	207	5	I_UNDO1
SYSTEM	212	5	I_OBJ1
SYSTEM	217	5	I_OBJ2
SYSTEM	222	5	I_IND1

Figure 10.4*Tablespace fragmentation.*

By examining the blocks, you can determine whether there are any segments missing—a process that is quite time consuming and tedious. Another way to detect fragmentation is by using one of many third-party monitoring and defragmentation tools.

One way to eliminate fragmentation is to export the table or tablespace data, remove and re-create the table or tablespace, and import the data. Although this process fixes the problem at hand, it may not solve the problem that caused the fragmentation to occur in the first place.

You can prevent fragmentation by properly sizing segment storage parameters so that your tables don't span excessive numbers of extents. It can also help to group segments with similar growth characteristics in the same tablespaces. It is also beneficial to avoid unnecessary table or index drops, which also cause fragmentation. Probably of most importance is to separate temporary segments into their own tablespaces.

By eliminating fragmentation, you can reduce excessive I/O and CPU usage, streamlining data accesses. And when you reduce any overhead and unnecessary I/O, you improve system performance.

Hash Clusters

A *hash cluster* is similar to a cluster but uses a *hash function* rather than an index to reference the cluster key. A hash cluster stores the data based on the result of a hash function. The hash function is a numeric function that determines the data block in the cluster based on the value of the cluster key. Figure 10.5 shows a hash cluster.

Figure 10.5

A hash cluster.

DOGS Hash Cluster		
250	1	Border Collie
	Dash	Jones
	Chip	Jones
	Piper	Turner
	Pierce	King
	Toller	King
2	Sheltie	
	Duncan	Miller
	Spoils	Ward
3	Jack Russell Terrier	
	Rugby	Smith
	Ruthie	Smith
251	4	Golden Retriever
	Splash	Blake
	B.J.	Allen
5	All American (Mix)	
	Shasta	Adams

To find the data block in an index cluster, there must first be one or more I/Os to the cluster index to find the correct data block. In a hash cluster, the cluster key itself tells Oracle where the data block is, an arrangement that can reduce to one the number of I/Os needed to retrieve the row.

In contrast to the index cluster, which stores related data together based on the row's cluster key value, the hash cluster stores related rows together based on their hash values.

The number of hash values is determined by the `HASHKEYS` value parameter of the `CREATE CLUSTER` command. The number and size of the cluster keys are very important and should be carefully calculated.

Do not use hash clusters on tables where table scans are often performed on only one of the tables in the cluster. The additional space required by the cluster and additional I/O can reduce performance.

Do not use a hash cluster on a table where the application frequently modifies the cluster key or when the table is constantly being modified. Because the cluster key is based on a calculation, significant overhead is involved in constantly recalculating the key.

When To Hash

Although hash clusters can be used in a similar fashion to index clusters, you do not have to cluster the tables. In fact, it is frequently useful to create a single table as a hash cluster to take advantage of the hashing feature. By using hashing, you may be able to retrieve your data with only one I/O rather than the multiple I/Os required to retrieve data using a B*-tree index.

Because hashing uses the value of the data to calculate the data block in which the desired data is located, hashing is best used on tables that have unique values for the cluster key and where the majority of queries are equality queries on the cluster key. For equality queries, the data is usually retrieved in one read operation; the cluster key does not have to be a single column. If the typical query uses an equality on a set of columns, use these columns to create a composite key.

Hashing is also most optimal when the table or tables are fairly static in size. If the table stays within its initial storage allocation, hashing usually does not cause a performance degradation. But if the table grows out of its initial allocation, performance can degrade because overflow blocks are required.

Hashing may degrade the performance of table scans because the hashing process reads blocks that may not have much data in them. Because the table is originally created by laying out the data into the cluster based on the value of the cluster key, some blocks may have few rows.

Hashing can also degrade performance when the value of the cluster key changes. Because the location of the block in which the data resides is based on the cluster key value, a change in that value can cause the row to migrate in order to maintain the cluster.

An good candidate for hashing has the following properties:

- ◆ The cluster key value is unique.
- ◆ The majority of queries are equality queries on the cluster key.

- ◆ The size of the table is static; very little growth occurs.
- ◆ The value of the cluster key does not change.

An example of a good hashing candidate is a table used for storing parts information. By using a hash cluster keyed on the part number, access can be extremely efficient and fast. Any time you have a somewhat static table with a unique column value or set of column values, consider creating a hash cluster.

Just as with index clusters, there are both advantages and disadvantages in using hash clusters. Hash clusters are efficient in retrieving data based on equality queries on the cluster key. If you are not retrieving data based on that key, the query is not hashed. As with the index cluster, you see a performance decrease when executing `INSERT` statements in a hashed table.

With both index clusters and hash clusters, make a careful determination about whether a cluster can help performance based on the access patterns on the tables. As with many aspects of RDBMS, tuning based on a wrong decision can end up costing performance.

If you can take advantage of hashing by meeting somewhat strict criteria, you can see very good performance. Hashing is extremely efficient if you can meet the criteria just described.

Indexes

An *index* is an optional structure designed to help you achieve faster access to data. Just like the index in this book, an Oracle index is logically and physically independent of the data in the associated table or cluster. You can use the index to speed access to the data or you can retrieve the data independently from the index by searching the tables for it. When optimally configured and used, indexes can significantly reduce I/O to the data files and greatly improve performance.

The presence of an index is transparent to the user or application and requires no application changes. However, if you are aware of an index, you may be able to take advantage of it when forming SQL statements (this topic is discussed in Chapter 26, “Tuning SQL Statements”). The only indication of an index may be an improved access time to data.

Once an index has been created for a table, Oracle automatically maintains that index. Inserts, updates, and deletions of rows in the table automatically update the related indexes.

A table can have any number of indexes, but the more indexes there are, the more overhead is incurred during table updates, inserts, and deletions. This overhead is incurred because all associated indexes must be updated whenever table data is altered.

TIP: If you use Oracle version 7.1 or later, you can create an index with the Parallel Index Creation feature of the Parallel Query option. Using this feature greatly reduces index creation time.

Index Types

There are several different types of indexes. An index can be limited to one column value or can consist of several columns. An index can be either unique or nonunique.

A *unique index* is an index value that has the additional constraint that it cannot be duplicated. Although this constraint may be specified, it is usually better to associate this constraint with the table itself rather than with the index. Oracle enforces UNIQUE integrity constraints by automatically defining a unique index on the unique key.

A *nonunique index* does not impose the constraint that the index value be unique. Such an index can be quite useful when quick access is desired on a nonunique value.

Another type of index is a *composite index*, an index that indexes several columns in a table. These column values can be in any order and the columns do not have to be adjacent in the table.

A composite index is useful when SELECT statements have WHERE clauses that reference several values in the table. Because the index is accessed based on the order of the columns used in the definition, it is wise to base this order on the frequency of use. The most-referenced column should be defined first, and so on.

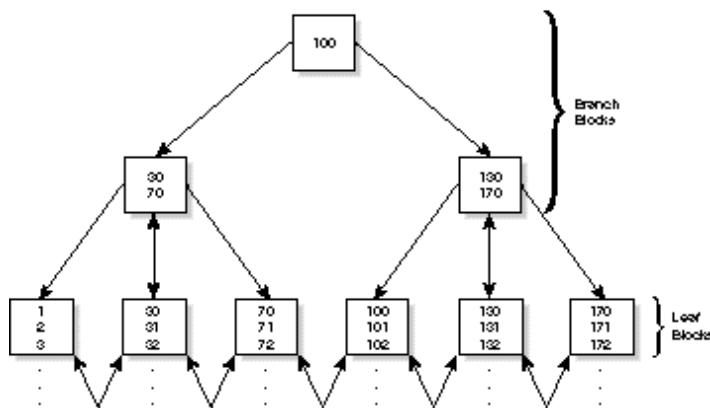
The index should be created based on the values accessed in the application; the application should be developed to take advantage of these indexes. Having knowledge of and influence over these indexes can be very useful to the application developer.

How the Oracle Index Works

When an index is created, an index segment is automatically allocated. This index segment contains information that speeds access to data by determining the location of indexed data with as few I/Os as possible. Oracle indexes data by using an index structure known as a *B*-Tree index*.

A B*-Tree index is designed to balance the access time to any row. A B*-Tree index is a tree of descending comparison values (see Figure 10.6). As you traverse down the index, you compare the desired value with the values in the upper-level index blocks called *branch blocks*. Based on the outcome of the comparison with the branch blocks, you compare the desired value with more branch blocks until you reach the lowest-level index blocks. The index blocks on the lowest level, called *leaf blocks*, contain every indexed data value and the associated ROWID of that data.

With a unique index, there is one ROWID per data value in the leaf block (see Figure 10.7). With a nonunique index, there may be several values associated with the data value. In the case of the nonunique index, the data values are sorted first by the index key and then by the ROWID.

Figure 10.6*The B*-Tree index structure.***Figure 10.7***The index block structure.***Index Block**

VALUE	ROWID
130	XXXX
131	XXXX
132	XXXX
133	XXXX
134	XXXX

With a B*-Tree index, all the leaf blocks are at the same level. Access of index data takes approximately the same time regardless of the value of the data. B*-Tree indexes provide quick access to data whether it is an exact match or a range query. In addition, B*-Tree indexes provides good performance regardless of the size of the table—and the performance does not degrade as the table grows.

What To Index

An index is effective only when it is used. The use of the index is mostly determined by the column values that are indexed. Remember that the more indexes you have on a table, the more overhead is incurred during updates, inserts, and deletes. Therefore, it is important to index selectively.

Use the following guidelines for deciding which tables to index:

- ◆ Index tables when queries select only a small number of rows. Queries that select a large number of rows defeat the purpose of the index. Use indexes when queries access less than 5 percent of the rows in the table.
- ◆ Don't index tables that are frequently updated. Updates, inserts, and deletes incur extra overhead when indexed. Base your decision to index on the number of updates, inserts, and deletes relative to the number of queries to the table.

- ◆ Index tables that don't have duplicate values on the columns usually selected in WHERE clauses. Tables in which the selection is based on TRUE or FALSE values are not good candidates for indexing
- ◆ Index tables that are queried with relatively simple WHERE clauses. Complex WHERE clauses may not take advantage of indexes.

If you decide to use an index, it is important to decide the columns on which you put the index. Depending on the table, you may choose to index one or more columns.

Use the following guidelines for deciding which columns to index:

- ◆ Choose columns that are most frequently specified in WHERE clauses. Frequently accessed columns can most benefit from indexes.
- ◆ Don't index columns that do not have many unique values. Columns in which a good percentage of rows are duplicates cannot take advantage of indexing.
- ◆ Columns that have unique values are excellent candidates for indexing. Oracle automatically indexes columns that are unique or primary keys defined with constraints. These columns are most effectively optimized by indexes.
- ◆ Columns that are commonly used to join tables are good candidates for indexing.
- ◆ Frequently modified columns probably should not be index columns because of the overhead involved in updating the index.

In certain situations, the use of composite indexes may be more effective than individual indexes. Here are some examples of where composite indexes may be quite useful:

- ◆ When two columns are not unique individually but are unique together, composite indexes may work very well. For example, although columns A and B have few unique values, rows with a particular combination of columns A AND B are mostly unique. Look for WHERE clauses with AND operators.
- ◆ If all values of a SELECT statement are in a composite index, Oracle does not query the table; the result is returned from the index.
- ◆ If several different queries select the same rows with different WHERE clauses based on different columns, consider creating a composite index with all the columns used in the WHERE statements.

Composite indexes can be quite useful when they are carefully designed. As with single-column indexes, they are most effective if applications are written with the indexes in mind.

Once you have created the index, you should periodically use the SQL Trace facility to determine whether your queries are taking advantage of the indexes. It may be worth the effort to try the query with and without indexes and then compare the results to see whether the index is worth the space it uses.

In summary, indexes can significantly improve performance in your system if they are used properly. You must first decide whether an index is appropriate for the data and access patterns in your particular system. Once you decide to use an index, you must decide which columns to index.

Indexing an inappropriate column or table can actually reduce performance. Indexing appropriately can greatly improve performance by reducing I/Os and speeding access times.

Careful planning and periodic testing with the SQL Trace feature can lead to a very effective use of indexes, with optimal performance being the outcome.

Multiblock Reads

When performing table scans, Oracle has the ability to read more than one block at a time, thus speeding up I/O. By reading more than one block at a time, Oracle reads a larger block from the disk and eliminates some disk seeks. By reducing disk seeks and reading larger blocks, both I/O and CPU overhead are reduced.

This feature is called *multiblock reads*. Multiblock reads are beneficial but take advantage of only contiguous blocks. Blocks in an extent are always contiguous. If your data is in many small extents, the effect of multiblock reads is reduced.

The amount of data read in a multiblock read is specified by the Oracle initialization parameter, `DB_FILE_MULTIBLOCK_READ_COUNT`. The value for this parameter should always be set high because there is rarely any disadvantage in doing so. The size of the I/Os depends on both `DB_FILE_MULTIBLOCK_READ_COUNT` and `DB_BLOCK_SIZE`.

To take advantage of multiblock reads, you should try to configure your system so that the database blocks are as contiguous as possible. To do this, you should try to create your database with optimally sized extents.

Creating these extents may not be a straightforward process, however. By creating extents too large, Oracle may have a difficult time finding enough contiguous space to create these extents. On the other hand, creating extents too small not only adversely affects multiblock reads, it also causes more dynamic extensions. Knowing what your initial data and growth patterns will be may help in sizing your extents.

Multiblock Writes

New in Oracle 7.3 is the multiblock writes feature. Multiblock writes are similar to multiblock reads and have many of the same requirements. Under certain conditions, you can perform multiblock writes:

- ◆ Multiblock writes are available through the direct path loader as well as through sorts and index creations. As with multiblock reads, multiblock writes reduce I/O and CPU overhead by writing multiple database blocks in one larger I/O operation.

- ◆ The amount of data written in a multiblock write is specified by the Oracle initialization parameter, `DB_FILE_MULTIBLOCK_WRITE_COUNT`. The size of the I/Os depends on both `DB_FILE_MULTIBLOCK_READ_COUNT` and `DB_BLOCK_SIZE`.

Parallel Query Option

The Oracle Parallel Query option makes it possible for some Oracle functions to be processed by multiple server processes. These functions are queries, index creation, data loading, and recovery. In each of these functions, the general principle is the same: keep the processing going while Oracle is waiting for I/O.

For most queries, the time spent waiting for the data to be retrieved from disk usually overshadows the amount of time actually spent processing the results. With the Parallel Query option, you can compensate for this by using several server processes to execute the query. While one process is waiting for I/Os to complete, other processes can be executing. If you are running on a Symmetric Multiprocessor (SMP) computer, a cluster, or an Massively Parallel Processing (MPP) machine, you can take maximum advantage of the Parallel Query option.

Many processes working together can simultaneously process a single SQL statement, a situation known as *parallel query processing*. The other functions are known as *parallel index creation*, *parallel loading*, and *parallel recovery*, each of which is discussed in the following sections.

Parallel Query Processing

Parallel Query Processing allows certain Oracle statements to be run in parallel by multiple server processes. The Oracle server can process the following statements in parallel:

- ◆ `SELECT` statements
- ◆ Subqueries in `UPDATE` and `DELETE` statements
- ◆ `CREATE TABLE tablename AS SELECT` statements
- ◆ `CREATE INDEX` statements

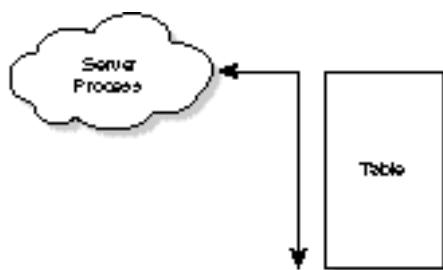
Parallel queries are effective on large operations such as table scans and sorts.

Parallel Query Operation

With traditional queries such as table scans, the server process reads the data sequentially (see Figure 10.8). Much of the time spent in this query is spent waiting for I/Os to complete.

Figure 10.8

A table scan without parallel query.

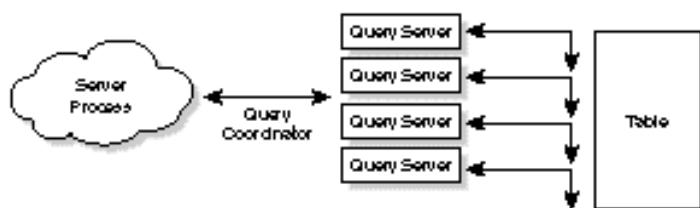


A parallel query splits the query into several different pieces, each one processed by a different server process. These processes are called *query servers*. The query servers are dispatched by a process known as the *query coordinator*.

The query coordinator dispatches the query servers and coordinates the results from all the servers to send back to the user. The result of this arrangement is that many smaller table scans take place under the hood (transparent to the user). From the user's standpoint, it is simply a much faster table scan. Figure 10.9 shows a parallel query.

Figure 10.9

A table scan with parallel query.



The query coordinator is given an SQL statement and a *degree of parallelism* and is responsible for dividing the query among the query servers and integrating the individual results into one result. The degree of parallelism is the number of query servers assigned to the particular query.

The Oracle server can make parallel the following operations:

- ◆ Joins
- ◆ Sorts
- ◆ Table scans

Each of these operations have requirements that determine how the query is parallelized. The performance achieved by the parallel query is determined both by the size of the data to be accessed and the degree of parallelism achieved.

How the query is parallelized (if at all) is determined by the query coordinator. The decision is made in this order:

1. The optimizer determines the execution plan of the statement.
2. The query coordinator determines which operations can be performed in parallel.

3. The query coordinator determines how many query servers to enlist.
4. The query coordinator enlists query servers that perform the query.
5. The query coordinator reassembles the resulting data and passes it back to the user.

The degree of parallelism is determined using the following precedence:

1. **Query Hints.** User-defined hints included in the SQL statement have the highest precedence.
2. **Table Definition.** The default degree of parallelism defined for the table has second precedence.
3. **Initialization Parameters.** Finally, the Oracle initialization parameters are used.

Regardless of what these values are set to, the number of query servers cannot exceed the number of query servers available in the query server pool. This number is specified by the Oracle initialization parameter, PARALLEL_MAX_SERVERS.

Hints for the degree of parallelism are set within a comment string in the SQL statement. The syntax of this comment is as follows:

```
PARALLEL ( alias_or_tablename , [ integer/DEFAULT ] [ , integer/DEFAULT ] )
```

The PARALLEL hint specifies the table or alias being scanned, followed by a value for the number of query servers to be used (or the DEFAULT). The final optional value specifies how the table is to be split among different instances of a parallel server. These hints are described in detail in Chapter 30, “Using Hints.” Here is an example using the DOGS table introduced earlier in this chapter:

```
SELECT /*+ FULL(dogs) PARALLEL(dogs, 4) */  
dogname  
FROM dogs;
```

When you add the FULL and PARALLEL hints to this statement, the Oracle optimizer creates an execution plan that uses a full-table scan. Furthermore, this tablescan is executed with a parallel degree of 4 if the query servers are available. This statement overrides both the degree of parallelism specified in the table definition and the default Oracle initialization parameters.

The hint NOPARALLEL disables parallel scanning of a table and overrides the specified degree of parallelism. The NOPARALLEL hint has the following syntax:

```
NOPARALLEL ( alias_or_tablename )
```

Parallel Query Tuning

Parallel query operations can be very effective on multiprocessor or parallel-processing computers; they can also be effective on uniprocessor systems where much of the time is spent waiting for I/O operations to complete. Systems with sufficient I/O bandwidth—and especially systems with disk arrays—benefit from parallel query operations.

If your system is typically processing at 100 percent of your CPU utilization and you have a small number of disk drives, you will probably not benefit from parallel query operations. If your system is extremely memory limited, you also will probably not benefit from parallel query operations.

The two areas that can be tuned for parallel queries are I/O and parallel servers. By properly configuring your data files, you can help parallel queries be more effective.

I/O Configuration

The function of a parallel query is to split up query operations so that they more effectively take advantage of the system. One of the ways a parallel query does this is by allowing the processing of the query to continue while pieces of the query operation are stalled waiting for I/Os to complete. Parallel queries are not effective if the entire table is limited to one disk drive.

By striping the table across many drives, I/Os can be distributed and a higher level of parallelism can occur. Striping can be done with OS striping, with Oracle striping, or (better yet) with a hardware disk array. Refer to Chapters 14 and 15 for detailed information on this topic.

Large contiguous extents can also help performance in parallel query operations. During scan operations, the query coordinator splits contiguous ranges of blocks into large, medium, and small groups of blocks. Each query server is given a large group of blocks to start with, progressively working its way down to the small group of blocks until the scan is completed. This is done to try to balance the load done by each query server. If there are several large extents in a table, the query coordinator can find blocks to dispatch to the query servers much more easily.

TIP: Remember to make your temporary tablespace up of several large extents on a striped volume. This arrangement helps sorting performance.

Degree of Parallelism

Properly distributing I/Os and the degree of parallelism are the two most important things to tune in the Parallel Query option. Tuning the degree of parallelism is partially trial and error and partially analysis. It is very important to take notes when you are experimenting with the degree of parallelism. Your first guess should be based on the following factors:

- ◆ The CPU capacity of your system. The number and capacity of CPUs have an effect on the number of query processes you should run.
- ◆ The capacity of the system to handle large numbers of processes. Some OSes can handle many simultaneous threads; others are more limited.
- ◆ The system load. If the system is already running at 100 percent capacity, the degree of parallelism doesn't have much effect. If you are running at 90 percent, too many query processes can overload the system.

- ◆ The amount of query processing on the system. If most operations are updates but there are a few critical queries, you may want many query processes.
- ◆ The I/O capacity of the system. If your disks are striped or if you are using a disk array, you should be able to handle a large number of parallel queries.
- ◆ The types of operations. Are you doing a lot of full-table scans or sorts? These operations benefit greatly from parallel query servers.

All these parameters should have some influence on the degree of parallelism you set up for your system. Remember that the preceding points are just guidelines to help with your best guess as a starting point. Here are a few other suggestions:

- ◆ CPU-intensive operations such as sorts should indicate a lower degree of parallelism. CPU-bound tasks are already taking advantage of the CPUs and tend not to be waiting for I/O.
- ◆ Disk-intensive operations such as full-table scans should indicate a higher degree of parallelism. The more operations waiting on I/O, the more the system can benefit from another query server.
- ◆ Many concurrent processes should indicate a lower degree of parallelism. Too many processes can overload the system.

Once you determine your starting point, you can monitor your system by querying the dynamic performance table, V\$PQ_SYSSTAT. You can do this with the query shown here:

```
SQL> select * from v$pq_sysstat;
```

STATISTIC	VALUE
Servers Busy	0
Servers Idle	12
Servers Highwater	16
Server Sessions	380
Servers Started	4
Servers Shutdown	4
Servers Cleaned Up	0
Queries Initiated	21
DFO Trees	77
Local Msgs Sent	2459361
Distr Msgs Sent	0
Local Msgs Recv'd	2459318
Distr Msgs Recv'd	0

13 rows selected.

When looking at the output from this query, you will find the following statistics quite useful:

- ◆ *Servers Busy*: This is the number of servers busy at any one time. Check this statistic several times to get a good idea of the average value. If the value is equal to the initialization parameter, PARALLEL_MIN_SERVERS, you have probably configured too many query servers.

- ◆ *Servers Idle*: This is the number of servers idle at any one time. If you always have many idle servers, consider reducing PARALLEL_MIN_SERVERS.
- ◆ *Servers Started*: The number of query servers that have started up in this instance. If the value for Servers Busy is low but you see a large number of Servers Started, you may be using query servers sporadically.
- ◆ *Servers Shutdown*: The number of query servers that have been shut down because they are idle. This value is most likely similar to the Servers Started value.

Once you determine your degree of parallelism, begin testing; evaluate the information you get from V\$PQ_SYSSTAT and from your operating system-monitoring facilities. Keep an eye out for CPU usage and excessive waiting for I/O. If the CPU usage is too high, try reducing the degree of parallelism. If the CPU usage is too low and there is significant waiting for I/O, try increasing the degree of parallelism.

Remember that the degree of parallelism is determined by SQL hints, table definitions, and initialization parameters. The total number of query servers is determined by the initialization parameter, PARALLEL_MAX_SERVERS; the number started up initially is determined by the initialization parameter, PARALLEL_MIN_SERVERS.

The total number of query servers in use is the number of queries executed in parallel multiplied by their degree of parallelism. If you try to use more than PARALLEL_MAX_SERVERS, you will not be able to parallelize your query.

Direct-Write Sorts

You can use the Direct Write Sort option with the Parallel Query option and have the query servers each perform their own direct writes.

As you have seen earlier in this chapter, using direct writes causes the server processes to write the output of sort operations directly to disk, bypassing the buffer cache. The effect of direct writes is that, for sort operations, large amounts of block buffers are not ejected from the buffer cache. This leaves the buffer cache available for normal queries and updates. When using direct-write sorts with the Parallel Query option, each query server gets its own set of direct-write buffers.

Remember that direct-write sorts take more memory than normal sorts. The amount of memory it uses with the Parallel Query option can be determined with the following formula:

```
Direct Write Sort Memory = ( Number of Query Servers ) * ( SORT_WRITE_BUFFERS ) *  
( SORT_WRITE_BUFFER_SIZE )
```

Only use direct-write sorts if you have sufficient memory and temporary disk space. The temporary disk space should have a sufficient I/O bandwidth to handle the load.

Parallel Index Creation

Another feature of the Parallel Query option is its ability to create indexes in parallel. With the parallel index creation feature, the time it takes to create an index can be greatly reduced.

Similar to parallel query processing, a coordinator process dispatches two sets of query servers. One set of query servers scans the table to be indexed to obtain the ROWIDs and column values needed for the index. Another set of query servers performs the sorting on those values and passes off the results to the coordinator process. The coordinator process then puts together the B*-tree index from these sorted items.

When creating an index, the degree of parallelism follows the same precedence as it does in parallel query processing. The first value used is an optional PARALLEL clause in the `CREATE INDEX` statement, followed by the table definition, and finally the initialization parameters.

Creating an index in parallel can be several times faster than creating an index by normal means. The same conditions apply for index creation as were given for parallel query processing. A system that has been configured to take advantage of parallel query processing will also see good performance from parallel index creation.

Parallel Loading

Loading can be done in parallel by having multiple concurrent sessions perform a direct path load into the same table. Depending on the configuration of the system, you can see excellent load performance by loading in parallel. Because loading is both CPU and I/O intensive, in an SMP or MPP environment with a high bandwidth I/O subsystem, you should see good results.

Parallel loads are performed by multiple direct loader processes each using the `PARALLEL=TRUE` and `DIRECT=TRUE` options. When you specify `PARALLEL=TRUE`, the loader does not place an exclusive lock on the table being loaded as it would otherwise. During the parallel load, the loader creates temporary segments for each of the concurrent processes and merges them together on completion.

Although parallel loading performs best when each temporary file is located on a separate disk, the increased performance of the load does not usually justify the complexity of the manual striping needed to do this. I still recommend striping the tables on an OS level—or preferably on a hardware disk array (refer to Chapter 15, “Disk Arrays”). Performance can be improved by putting each of the input files on a separate volume to take advantage of the sequential nature of the reads.

Parallel loading can be beneficial, especially if load time is critical in your environment. By putting each of the input files on separate disk volumes, you can increase performance. Overall, the general tuning principles used in parallel query processing are valid in parallel loading also.

Parallel Recovery

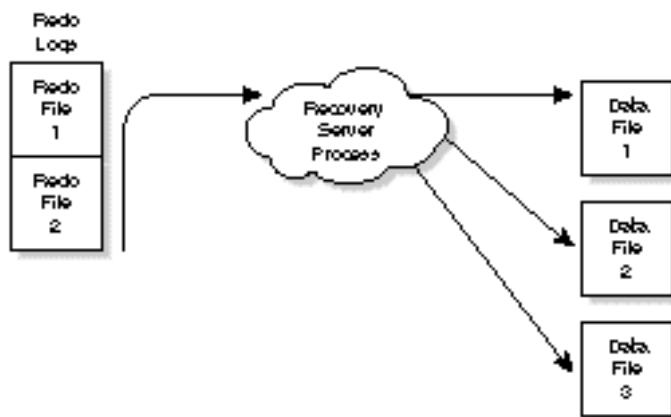
Parallel recovery is probably my favorite feature of the Parallel Query option. When benchmarking Oracle and testing hardware and software, it is often necessary to intentionally crash the system to prove recoverability. With the Parallel Recovery option, the time it takes to perform and instance recovery can be dramatically reduced.

Recovery time is significantly reduced when the system being recovered has many disks and supports Asynchronous I/O. For a small system that has few drives or for an operating system that does not support Asynchronous I/O, it may not be wise to enable parallel recovery.

In traditional recovery, one process both reads from the redo log files and applies changes to the data files (see Figure 10.10). This operation can take a significant amount of time because the recovery process must wait for disk I/Os to complete.

Figure 10.10

Traditional recovery.

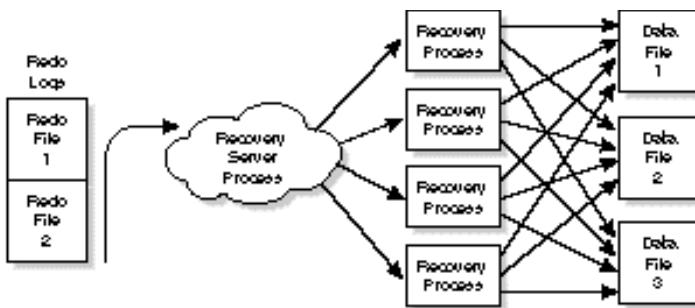


With the Parallel Recovery option, one process is responsible for reading and dispatching redo entries from the redo log files and passing those entries on to the recovery processes that apply the changes to the data files (see Figure 10.11).

Because the dispatcher process reads from the redo log files sequentially, the I/O performance is much higher than that of the recovery processes that are writing random data throughout the data files. Because writing the data is very seek intensive, it is a good idea to have one or two recovery processes for each data disk in the system.

By having more recovery processes, you can have more outstanding I/Os and thus use all the data drives simultaneously. Because recovery is done at instance startup, this arrangement reduces dead time when no other database processing can be done.

The number of concurrent recovery processes is set with the initialization parameter RECOVERY_PARALLEL. The value of this parameter cannot exceed the value specified in the initialization parameter PARALLEL_MAX_SERVERS.

Figure 10.11*Parallel recovery.*

By specifying a sufficient number of recovery servers, you will see an immediate improvement in instance recovery time. Do not use parallel recovery if your system does not support Asynchronous I/O or you are limited to a small number of disk drives. If your I/O subsystem has a high bandwidth and your data is properly striped (either through software or hardware), you should see very good improvement.

In summary, the Parallel Query option is useful in distributing processing loads so that CPUs are kept busy processing while other processes are waiting for I/Os to complete. With multiprocessor machines, the Parallel Query option can be quite beneficial; this is not to say that the option is not beneficial on uniprocessor machines as well.

NOTE: Probably the biggest performance problem I have come across is a lack of disk drives. As larger and larger disks are produced at lower and lower prices, many installations end up with I/O problems caused by a lack of disks (see Chapters 14 and 15). The Parallel Query option can help only in systems where I/O is not a bottleneck. When I/O is not a problem, you will see significant gains from parallel queries.

If you have processes waiting for queries to complete and a sufficient number of disk drives, you will see an improvement with parallel queries, regardless of whether you are on a multiprocessor or uniprocessor system.

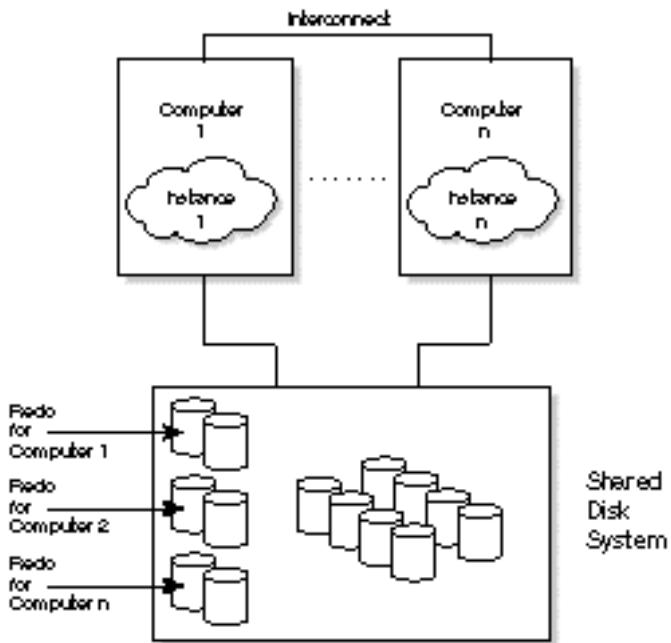
Parallel Server Option

The Parallel Server option is one of the most innovative and impressive options available from Oracle. With the Parallel Server option, you can cluster several computers together using a shared-disk subsystem and have multiple Oracle instances access the same database (see Figure 10.12). If your application is suitable, you can see very good scalability by adding additional computers.

The Oracle Parallel Server option uses a sophisticated locking mechanism in conjunction with a shared-disk subsystem to allow multiple instances to access the same data. Communication

between the computers is done through a *server interconnect*, which usually consists of high-speed network accesses at a very low level. Using the traditional network stack does not provide the performance required for a cluster interconnect.

Figure 10.12
A Parallel Server configuration.



The server interconnect provides two functions: to communicate locking information and to provide a system heartbeat. The system heartbeat communicates to other systems in the cluster that it is still operational. If the heartbeat message does not arrive, other servers in the cluster assume that the system is nonfunctional and roll back transactions that have not been committed.

Locking is performed with a process called the Distributed Lock Manager (DLM). The DLM is responsible for locking data that is being modified so that the data cannot be modified in another instance. Locking ensures data integrity across the entire cluster. A data block or group of blocks is locked until such time as another instance needs that data.

Because the locks are held until needed by another instance, if you can partition your users so that users accessing data in a particular table all use the same instance to access that data, you will have reduced lock contention. You can enhance performance by carefully partitioning the data and the users. If you partition the data into update-intensive and read-intensive tables, you will also benefit.

At instance startup, a number of Parallel Cache Management (PCM) locks are created. PCM locks are used to lock data blocks being accessed within each instance to guarantee that multiple instances do not alter the same data.

You can use PCM locks to lock data blocks for reading or for updating. If a PCM lock is used as a read-lock, other instances can acquire read-locks on the same data blocks. It is only when updating that an exclusive lock must be acquired.

PCM locks are allocated to data files; as such, they give you some flexibility over the configuration of the locks. A PCM lock locks one or more data blocks, depending on the number of PCM locks allocated to the data file and the size of the data file. Because there is an inherent overhead associated with PCM locks, it is not beneficial to overconfigure the locks.

If you know your data-access patterns, you can configure your system based on these general rules:

- ◆ **Partition work between servers.** Try to balance the systems so that users accessing the same table reside on the same computer. This arrangement reduces lock contention between machines. By segmenting the work, you can reduce the amount of lock traffic. Remember that once a lock is acquired, it is released only when another system needs to lock that data.
- ◆ **Put lots of PCM locks on tables with heavy update traffic.** If you have lots of updates, you can benefit from lowering the blocks-per-lock ratio. By increasing the number of locks, you increase overhead—but by having fewer blocks per lock, you may cut down on the percentage of locks with contention.
- ◆ **Use PCTFREE and PCTUSED to cause less rows per block on high-contention tables.** By doing this and decreasing the number of blocks per lock, you reduce the lock contention—at the cost of more locks and more space required.
- ◆ **Put fewer locks on read tables.** If you have tables that are mostly read, use fewer PCM locks. Read locks are not exclusive; the reduction in locks cuts down on interconnect traffic.
- ◆ **Partition indexes to separate tablespaces.** Because indexes are mostly read, you can benefit by needing fewer PCM locks. By segmenting the tables, you can put fewer PCM locks on the index tables and more on the data tables.

The dynamic performance tables V\$BH, V\$CACHE, and V\$PING contain information about the frequency of PCM lock contention. By looking at the FREQUENCY column in these tables, you can get an idea of the number of times lock conversions took place because of contention between instances.

The dynamic performance table, V\$LOCK_ACTIVITY, gives information on all types of PCM lock conversions. From this information, you can see whether a particular instance is seeing a dramatic change in lock activity. An increase in lock activity may indicate that you don't have a sufficient number of PCM locks on that instance. With this information, you can use the V\$BH, V\$CACHE, and V\$PING tables to identify the problem area.

The Parallel Server option can be effective if your application is partitionable. If all the users in your system must access the same data, a parallel server may not be for you. But if you can partition your workload into divisions based on table access or if you need a fault-tolerant configuration, the Parallel Server option may be for you.

If you use the Parallel Server option, you must take special care to properly configure the system. By designing the system properly, you can take maximum advantage of the parallel server features.

Spin Counts

Multiprocessor environments may benefit by tuning the parameter `SPIN_COUNT`. In normal circumstances, if a latch is not available, the process sleeps for a while and wakes up to try the latch again. Because a latch is a low-level lock, a process does not hold it very long.

If you are on a multiprocessor system, it is likely that the process holding the latch is currently processing on another CPU and will be finished in a short time. By setting `SPIN_COUNT` to a value greater than zero, the process spins while counting down from `SPIN_COUNT` to zero. If the latch is still not available, the process goes to sleep.

The time it takes to perform one spin count depends on the speed of your computer. On a faster machine, one spin count is quicker than one spin on a slower machine. However, the other process that is holding the resource will also be processing at a faster rate. The spin function itself only executes a few instructions, and is therefore very fast.

Enabling spin counts has some good points and some bad points. You may increase CPU usage by setting `SPIN_COUNT` and thus slow the entire system. Because sleeping is a somewhat expensive operation, sometimes you get the latch in fewer CPU cycles than the sleep-and-wake-up cycle would take and thus improve performance.

If the process spins and then must go to sleep anyway, there is definitely a performance loss. By carefully monitoring the system, you may be able to find an optimal value for `SPIN_COUNT`.

Monitor the statistics `MISSES` and `SLEEPS` in the dynamic performance table `V$LATCH`. If the `SLEEPS` value goes down, you are on the right track. If you never see any sleeps, it is not even necessary to tune `SPIN_COUNT`.

NOTE: The `SPIN_COUNT` initialization parameter is not available on all operating systems. Check your OS-specific documentation to see whether it is available on your system.

Summary

This chapter discussed many concepts and ideas you can use to further enhance performance. Chapter 9, “Oracle Instance Tuning,” was interested in instance tuning in general—things that always applied. This chapter began to look at more specific topics that may or may not apply, depending on your configuration. In fact, some of the things described in this chapter can hurt performance if they are not handled correctly.

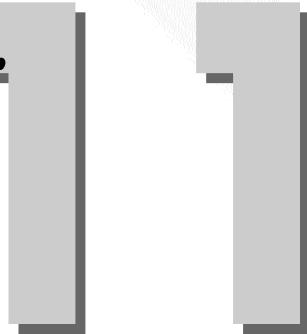
Clusters and indexes are very useful when the application can take advantage of them. In update-intensive applications where indexes are used, indexes can degrade performance because they must be updated along with the data. Similarly, clusters can be beneficial if the data access patterns use the cluster, else they can be a burden.

The Oracle Parallel Query option and the Parallel Server option can offer significant performance benefits. The Parallel Query option provides a mechanism to improve system utilization by splitting queries into multiple processes. The Parallel Server option provides a robust scaleable clustering option that can increase both performance and provide instantaneous failover capabilities. Your applications and needs will determine whether these options can benefit you.

When tuning a system, it is important to remember the risks you are taking. The last chapter discussed different tuning topics such as shared pool size and database block buffer size. Increasing the size of the shared pool and block buffers can increase cache hits; it also has very little risk of hurting performance. However, some of the topics addressed in this chapter may hurt performance if they are not configured correctly. You must assess the risks involved whenever you reconfigure, especially if this reconfiguration is going into production without intensive testing.

If you think about what you are changing and what those changes will affect—and if you understand the data access patterns of your applications—you minimize the risks. Think about how the changes you are making will affect the system. Try to understand how your applications will react. By doing so, you minimize risks and optimize results.

Chapter



Tuning the Server Operating System

The server operating system functions as a host for the RDBMS. The OS performs such tasks as providing access to data stored on disk, either through the file system or directly through the raw interfaces (available on some OSes). The OS also provides the network interface that SQL*Net uses to communicate to other machines. Other functions provided by the operating system include system performance monitoring and backup and recovery functions.

Although these functions are essential to the operation of Oracle, they should be done with as little system overhead as possible. The I/O and network functions can be configured to be streamlined to reduce overhead. As much as possible, the system should get out of the way and let Oracle do its work.

This chapter looks at the goals for tuning the server operating system and what areas are normally tuned. The next chapter looks at the specifics of tuning several different operating systems.

Goals

The goals for tuning the server operating system can be combined into a single goal: provide Oracle with the functions and abilities needed to operate in an optimal manner. The functions the OS provides are things such as a sufficient amount of memory, an optimal path to I/O, and the ability to create a sufficient number of processes to connect the desired number of users.

One goal for every operating system is to reduce the amount of overhead used by the OS. Many times, you may never need and never use some of the processes and drivers configured into the system. By removing these unnecessary processes (and freeing the resources they have allocated), you can reallocate these resources where they can be more effective—mainly, the Oracle SGA.

By default, the server OS may be tuned for file or print services and therefore allocate large amounts of memory for disk and print caching. By reducing the amount of memory allocated to these tasks, more memory can be used by Oracle.

Check to see what is turned on in the operating system by default. You may find many completely unnecessary processes or threads. If your system operates in a single-protocol network, it is unnecessary to run multiple protocols on the server. Any reduction in memory or processing overhead can be exploited by Oracle.

Other goals for tuning the server operating system are to provide the most optimal I/O path, reduce network contention, and configure OS features (such as Asynchronous I/O and post-wait semaphores) that may improve the performance of Oracle.

By providing an optimal path for I/O, you may see significant gains because most data access involves disk I/O. Anything you can do to improve I/O—such as reducing overhead, offloading non-Oracle work, and reducing fragmentation—results in significant performance gains. I/O is a critical area in any OS because it is typically the slowest component in any data access.

By reducing network contention, you may shorten latencies that cause performance problems. When I/Os are critical to transactions (such as with distributed databases), network contention can significantly hurt performance. If you can reduce overhead, you will increase performance.

Use any additional features in the OS that help improve the performance of Oracle—such as Asynchronous I/O and the post-wait semaphore—whenever possible. Although a performance gain by one single component may be insignificant, when you add it to many other small gains, you may see a large gain.

This chapter looks at many of the features you can enable in the server operating system. You will see that, regardless of the operating system, many of the OS functions and features are

similar—and many features differ. In Chapter 12, “Operating System-Specific Tuning,” you see how these features are specifically implemented in several popular operating systems.

Processes

It is necessary to tune only the number of processes you need to support the required number of users. Remember that you must also add enough processes to accommodate the background processes. Although there is usually no problem with the number of processes allowed on a system, some operating systems cannot handle a large number of users.

To calculate the number of user processes needed on your system, take the maximum number of concurrent users you need to support, add an additional 10 for Oracle server processes, and add another 5 to 10 percent for additional OS processes that often need to run. If your system supports additional users for development or other nonproduction usage, add these numbers also.

For some operating systems, you may also have to tune the maximum number of processes allowed for a particular user. Because the Oracle user ID owns the Oracle server processes, the Oracle user must be able to support many processes. This includes the background processes as well as other administrative processes you may invoke during backups and so on. A good rule of thumb is to make the maximum number of processes per user equivalent to the number of Oracle users plus 25. This number allows for the server processes as well as any administrative tasks you might be running.

The priority of the process is also of importance. Although, in some cases, putting some processes at a high priority can benefit performance, priority adjustment is a very tricky thing. I feel that increasing the priority of Oracle processes usually is not worth the tremendous effort needed to do it.

Increasing the priority of some processes may prevent other processes from getting enough CPU time. And starving some processes may cause the performance of the RDBMS as a whole to suffer.

One area in which adjusting priority may help without too much risk is in loading the database. If the system is not being used for user processing, you may benefit from raising the priority of the direct loader processes. The higher priority may keep the processes from being preempted, resulting in higher throughput. However, do this at your own risk.

CAUTION: I do not recommend adjusting the priority of any process. I don't feel that the benefit you can get outweighs the possible performance degradation you may see.

By putting any process at a high priority, you run the risk of not being able to stop it. If you put any Oracle task at very high priority, make sure that you have a user process of equal or higher priority that can stop it in the event of a runaway process.

Memory

Memory is the most important area you can tune in the server operating system. By reserving a sufficient amount of memory for the proper size SGA for your system, you will see a good balance between CPU and I/O. If your SGA is too small, you may see idle CPU cycles when Oracle is doing nothing but waiting for I/Os to complete. If this occurs, your machine has become I/O bound. Frequently, all you need is a larger SGA to achieve the right balance.

In NetWare and Windows NT, the memory used for Oracle is the same as any other memory used in the system. In the UNIX operating system, you must reserve *shared memory* for the SGA. The shared memory area is a special type of memory that allows multiple processes to access the same memory.

In some UNIX implementations, you may be able to take advantage of Page Size Extension (PSE) memory. PSE memory uses a much larger page size and thus reduces memory page lookup thrashing in the OS. Be careful of the restrictions associated with PSE memory; the amount of PSE memory you allocate is frequently reserved at boot time and is not available for other uses in the system, even if it is not used.

I/O

I/O is another important area you must sometimes tune in the server OS. In some operating systems, there is nothing to tune; in others, you can change parameters to improve I/O performance. Depending on the operating system, you may have to change the database block size to take advantage of the OS block size.

Of course, the best way to improve I/O performance is by carefully distributing data files to take advantage of the available I/O bandwidth. By separating sequential and random I/Os, you can take advantage of the available disk throughput, as described in Chapter 13, “System Processors,” and Chapter 14, “Advanced Disk I/O Concepts.”

If you use OS striping for fault tolerance or performance, you may have to tune several parameters within the disk striping or file system subsystems. This requirement varies depending on the operating system and striping method.

The way in which operating systems do I/O is also very important. Although the operating systems examined in Chapter 12, “Operating System-Specific Tuning,” have various methods of handling I/Os, many concepts are the same. Whether you use Direct FS under Windows NT or NetWare, raw UNIX synchronous I/O, or raw devices, the way Oracle does I/O must be handled the same. And regardless of the operating system you use, the concept of Asynchronous I/O is always the same.

I/O Methods

In order for Oracle to manage its cache (the SGA) and guarantee data recovery in the event of a system failure, OS disk caching must be turned off. If the Oracle processes believe that data has been written to disk, but in fact the data was in the OS disk cache at the time of failure, Oracle cannot recover that data. To avoid this, Oracle uses disk I/O interfaces that do not use the OS disk cache. The following interfaces are used:

Direct FS	With Novell NetWare and Microsoft Windows NT, the Direct FS interface is used. This interface uses the standard file system but does not cache the data in the OS disk cache.
Synchronous I/O	The UNIX equivalent of Direct FS. Disk I/Os are not cached. This interface is not related to the concept of Asynchronous I/O.
Raw devices	Raw devices are available under UNIX and Windows NT; they provide an interface to the I/O subsystem that avoids the file system. This interface is faster because it avoids unnecessary OS overhead.
Asynchronous I/O	With Asynchronous I/O, the process performing the I/O (usually the DBWR process) submits a list of I/Os to the OS and continues processing. When the I/Os are done, the process receives an interrupt and can go retrieve the data.

To have more insight into how the system operates, you should look a little deeper into how Oracle does I/O: Oracle has a sophisticated cache management system. Using the database block buffers and the shared pool, much of the frequently used data can be stored in memory for quick retrieval. Working closely with this cache management system is the Oracle redo log. With the cache and the redo log, Oracle has created both a high performing and robust database management system.

To keep the data files and redo log synchronized, Oracle must perform I/Os in a specific way. Oracle must make sure that when the RDBMS believes data has been written to the data files, that data is there. Oracle ensures this by using the operating system I/O interface. In NetWare or Windows NT, this interface is called the Direct FS interface. In UNIX, this interface is the synchronous I/O interface. In UNIX or Windows NT, you can also use raw disk devices.

Direct or Synchronous I/O

With direct or synchronous I/O, the process requesting the I/O must block on the physical I/O. To *block on the I/O* means that the process goes to sleep waiting for the I/O to return. With normal I/O, the process blocks on the I/O, but the I/O returns completed when the data has been written into the OS disk cache. With synchronous I/O, the process blocks on the I/O, but the I/O does not return until the data has been physically written to the disk. This method of I/O ensures data integrity.

Synchronous I/Os are so important because of the way Oracle does data recovery. Remember the concept of the *checkpoint*. At one point in time, Oracle writes out all the dirty buffers in the SGA. All the data that has been changed up to the time at which the checkpoint started is written out. Once the checkpoint has completed, the data files are updated to indicate that the data file is current up to the last checkpoint. Once Oracle has written data to the disk, no recovery is performed on that data in the event of a system failure. If Oracle thought it had written this data to disk, but in fact the data been written only to the OS disk cache, when a failure occurs, this data cannot be recovered and the database will have incorrect data in it. From the standpoint of the DBWR process, the data has been written to disk. Data integrity cannot be maintained without Direct FS or synchronous I/Os.

However, by using synchronous or direct I/O to ensure that all data has really been written to disk, there is no integrity problem. Only by using synchronous or direct I/O can data integrity be ensured. Every operating system offers some sort of direct or synchronous I/O.

Asynchronous I/O

Although the terms *asynchronous I/O* and *synchronous I/O* may seem to have similar meanings, they operate on completely different levels. In fact, the asynchronous I/Os used by Oracle must be synchronous.

With asynchronous I/O, the process requesting the I/O submits the I/O request and moves on. The OS response returns later. This arrangement allows a process (such as the DBWR process) to handle many outstanding I/Os at once without having to wait for each one to return. By using asynchronous I/Os, the DBWR can continue processing, submitting I/Os and retrieving the results of the I/Os later. With asynchronous I/O, the process submitting the requests is responsible for keeping track of all the I/Os it has outstanding.

NOTE: When I say that “asynchronous I/Os are in fact synchronous,” what I mean is that the OS does not tell the process that the I/O has completed until the I/O has actually been written to the physical medium. Therefore, the asynchronous I/O uses synchronous writes.

Miscellaneous

Each OS has its own unique features and operational methods. Over the years, many new features have been used to improve the performance of Oracle in different ways on different OSes. Some of these features and improvements have similarities; others are unique to the specific operating system.

Post-Wait Semaphore

With some varieties of the UNIX operating system, it was found that during the course of normal processing, at the beginning of a clock tick, various processes would try to acquire a particular resource; after trying unsuccessfully for a while to get the resource, the processes would go to sleep. Eventually, every process would go to sleep except the one holding the resource. When that process was done (if it did not have anything else to do or if it submitted an I/O), it also would go to sleep. Because sleeping processes wake up only on the clock tick, the last portion of the clock cycle wound up with no processes running.

Oracle and some of the UNIX vendors got together to solve this problem by inventing a device called a *post-wait semaphore*. When this new type of semaphore is available, it replaces the use of standard semaphores. By using this semaphore, Oracle has more control over the UNIX sleep and wakeup routines: a process that is freeing a resource can wake up any processes that are waiting (sleeping) for that resource.

Scheduling and Preemption

Another area you may be able to influence is tuning the OS scheduler and preemption system to better match the needs of Oracle. Some OSes are designed for the general case and must be tuned to provide the highest level of performance for more specific cases. This is true with how the typical OS does scheduling and preemption.

Here is how the standard timesharing model of scheduling works: A process is scheduled based on its priority. The process's priority may degrade over time based on how much CPU time it has already had or other factors. When a process is scheduled, it runs until some criteria is met. Here are some typical events that cause a process to relinquish the CPU:

- ◆ The process has finished all it needs to do.
- ◆ The process executes an I/O instruction that causes the process to go to sleep.
- ◆ The process has run for the maximum time slice interval.
- ◆ An interrupt occurs, which usually causes the process to be temporarily halted while the interrupt is serviced.
- ◆ The process is preempted by another process.

The final situation—when the process is preempted by another process—is most interesting to the database performance engineer. An OS designed for general processing makes some assumptions that are not necessarily true for all cases. One of these assumptions is that a lower-priority process should always be preempted by a higher-priority process.

Although this assumption may be true for long-running processes that never relinquish the CPU, rarely does an Oracle process run for very long before relinquishing the CPU voluntarily because of an I/O request.

When the process is preempted by a higher-priority process, it is most likely that the preempted process would have relinquished the CPU very soon anyway. What happens is that an unnecessary preemption occurs, wasting CPU and I/O resources.

To limit the waste of CPU and I/O resources caused by unnecessary preemption, several OS vendors have included tunable parameters that allow you to control preemption more closely. In many cases, you can turn off preemption and achieve greater performance by reducing process switches.

Cache Affinity

Cache affinity is another area that has received a lot of attention over the last few years. *Cache affinity* is a term used in multiprocessor OSes and refers to the act of trying to run a process on the same processor it last ran on. The theory behind cache affinity is that if there is any data from that process left in the CPU cache, running the process on the same cache results in a big win. For general-purpose computing, cache affinity may be a good thing.

However, when you are running a multiprocessor system with many users and heavy access patterns (such as a large OLTP system), it is extremely unlikely that any data is left in the CPU cache when it is time for your process to run again. What cache affinity does for a large multiprocessor system like this is to waste many CPU cycles in the OS scheduler trying to relocate a process for no good reason.

On the other hand, if you run a large application with a small number of processes (such as a DSS system), you may benefit from cache affinity. You will also see a benefit from cache affinity if you have a fair number of processes all running the same shared code.

Depending on your application and workload, you may find that many innovative OS features are either a benefit or extra overhead. By tuning only those features that are beneficial, and turning off those features that are just extra overhead, you may see some performance gains.

Summary

This chapter gave you an idea of the general areas that need attention in the server operating system. The next chapter looks specifically at several different server operating systems and the parameters that need attention in each one. Most of these operating systems are quite different from each other both in function and in tuning philosophy.

Because some OSes are designed to require very little tuning, they are very difficult to tune even when they need tuning; other OSes are designed to be very configurable and can be customized for each installation. Because Oracle runs on over 90 different platforms, the operating system you chose should be based on your own criteria.

Chapter **2**

Operating System-Specific Tuning

This chapter looks at the tunable parameters in specific operating systems. In this chapter, you see how to tune the operating system parameters discussed in the last chapter (there is an overview of the parameters and areas in the OS you can modify).

Many factors should lead you to choose a particular operating system to run in your installation. I have no intention of influencing that decision. Usually, your choice of operating system depends on how it fits into your installation in terms of connectivity and the administrative expertise you already have.

Each operating system has advantages and disadvantages, both in terms of ease of use and configurability. Sometimes, there is a tradeoff between the two. In this chapter, I try to represent the advantages and disadvantages of each operating system along with the tuning methodologies.

NOTE: Several popular operating systems are described in alphabetical order in this chapter. I have tried to pick the OSes that are typically used in the client/server market. I apologize if your particular OS is not included here. The chapter looks at NetWare first, then at Windows NT and OS/2, and finishes up with the UNIX operating system.

The goals of this chapter are to provide you with a basic understanding of what is available in the server operating system arena. Each operating system is unique, both in how the system operates and in the level of configuration available. By understanding something about the architecture of the OS, you may gain some insight into how you can best configure your system for Oracle.

NetWare

NetWare is the most popular file and print server operating system on the market. NetWare has grown in popularity over the last few years and is now seen as both a file-and-print server as well as a stable and robust application server. Recently, Novell has released an SMP version of NetWare. NetWare itself has always had very good performance as a single-processor operating system, but with the introduction of NetWare SMP, you can take advantage of scaleable performance increases by adding additional CPUs.

NetWare operates in a client/server configuration only. There is no facility within NetWare to provide login functionality from terminals. From the very beginning, NetWare was designed to support clients through network connections. NetWare is designed to be a lean-running operating system; the overhead associated with it is very small.

Architectural Overview

NetWare runs on Intel PC servers and provides file and print services to clients over network connections. The Novell network protocol of choice is SPX/IPX, but TCP/IP is also supported (the choice of network protocol usually depends on the protocol used by the PC clients).

Unlike the other operating systems described in this chapter, NetWare is not a virtual memory operating system. With NetWare, all the memory needed by Oracle and the server processes must fit into physical memory. The obvious disadvantage is that you must be careful not to exceed the bounds of the memory available. The advantage is that the system is guaranteed not to page and that all the overhead necessary to manage virtual memory is eliminated.

With NetWare SMP, the basic server functionality of the NetWare services is still done on the primary CPU. Certain applications, such as Oracle, have to be modified to take advantage of the additional CPUs. NetWare SMP operates in a master/slave mode.

The term *master/slave* refers to the way in which the multiprocessing architecture works. The base processor is the master and is responsible for dispatching work to the other CPUs, which operate as slave processors. The slave processors do not have the same software functionality as the master because the slaves do not run all the same processes as the master does.

The result of having a master/slave architecture is that the additional CPUs are not burdened by the overhead processing such as network connections and hardware interrupts that must be done by the master. The slave processors devote all their time to Oracle processing.

The slave processors run the Oracle background processes as well as the Oracle server processes. In this manner, an additional CPU is very scaleable. By adding a second CPU to your server, you may come close to doubling the performance of your system (if there is available I/O bandwidth).

Tuning Considerations

With NetWare, probably the most important tuning consideration is ensuring that you have sufficient memory available to run Oracle and the required number of server processes. Because NetWare does not use virtual memory, you must make sure that enough physical memory is available. Other areas of importance in tuning the NetWare system include both the network and I/O subsystem.

Memory

Memory is very important in the NetWare operating system. The initial design of NetWare focused on file and print services, which can be configured to use very little memory. NetWare is streamlined and can operate with a small amount of memory and very little overhead.

When you use NetWare as an application server, it requires much more memory. As you have seen in earlier chapters, effective use of the SGA can greatly affect performance. Because NetWare does not use virtual memory, it is much more important to have a sufficient amount of memory so that you do not exceed the available resources. Because you really want to have enough memory so that no paging occurs in virtual memory systems as well, this is not a problem. As long as there is sufficient memory, you will have no problems running Oracle on NetWare.

Reduce Unnecessary Memory Usage

You take some measures to reduce the amount of memory NetWare uses so that you can re-allocate that memory to Oracle. One way to reduce NetWare memory usage is to set MAXIMUM RECEIVE PACKET SIZE to the largest packed size supported by your protocol and Network Interface Card (NIC). By setting this value higher than the maximum supported by the protocol or

NIC, you waste space in memory by having underutilized packets. On the other hand, do not set this value too small, or you may waste CPU cycles by having to transmit more packets than necessary.

For Ethernet, try setting `MAXIMUM RECEIVE PACKET SIZE` to 1130; for 4 megabit-per-second Token Ring, set this parameter to 2154; for 15 megabit-per-second Token Ring, use 4202. Treat these values as a guideline. If the number of packets used increases, set these numbers higher.

Another way to reduce the amount of memory used by the OS is to set the parameter `VOLUME BLOCK SIZE` to the maximum of 64K. This parameter specifies the minimum block size that the NetWare file system can allocate to a particular file. This parameter does not affect Oracle because Oracle uses the Direct File Services (DIRECTFS) routines for disk access, but it does reduce the amount of memory NetWare uses for caching the directory structures. By reducing this value, more memory can be dedicated to Oracle.

The parameter `MAXIMUM RECEIVE PACKET SIZE` is located in STARTUP.NCF; the `VOLUME BLOCK SIZE` parameter is located in INSTALL.NLM. The NetWare parameter `CACHE BUFFER SIZE` is best left at the default of 4096 during run time, but boosting this value to 16384 during tablespace creation may help performance. The Oracle parameter `NW_FSTYPE` should be left set to DFS.

SGA Tuning

To maximize performance, allocate as much memory as possible to the SGA. Use the techniques discussed in Chapter 9, “Oracle Instance Tuning,” to determine whether memory will be best used for the shared pool or for database block buffers.

Unique to NetWare is the Oracle initialization parameter `NW_SGA_MAX_ALLOC`, which specifies the maximum size of each block (in kilobytes) in the SGA. The default value of `NW_SGA_MAX_ALLOC` is 256. This represents the largest block of contiguous memory Oracle can request from the NetWare memory pool.

In virtual memory systems, Oracle always gets contiguous memory (virtual memory); in UNIX, the memory area used by Oracle is contiguous. In NetWare, however, Oracle is not always guaranteed contiguous memory for the SGA. Because some structures in the SGA require contiguous memory, the `NW_SGA_MAX_ALLOC` parameter is necessary. The largest structure internal to the SGA that must be contiguous is `_DB_BLOCK_MULTIPLE_HASHCHAIN_LATCHES`; if Oracle cannot obtain a contiguous block of memory for this structure, it returns an `ORA00064` error message. If you see this message, you must increase the value of `NW_SGA_MAX_ALLOC` before you can start the Oracle instance.

User Capacity

Determining the amount of memory necessary for your application on a per-user basis is easy. Start up Oracle and note the amount of memory used by looking at the Resource Utilization

screen in NetWare's MONITOR.NLM. Specifically, look for the amount of memory in the Cache Non-Movable Memory pool. Once users start accessing the application in a typical manner, record the amount of memory again. Take the difference and divide that result by the number of users accessing the application. This result is the per-user memory usage. Multiply this value by the maximum number of users who will connect to the application to determine the amount of memory you must reserve for user connections. Be sure to leave a little extra memory just in case.

In addition to the memory required for each connection, there are a few other concerns if you are to support the required number of user connections. Part of the memory required for each user is the Program Global Area (PGA). The amount of memory used in the user process's PGA is limited by the initialization parameter `NW_PGA_MAX_ALLOC`. The default value of this parameter is `NW_SGA_MAX_ALLOC` and is usually sufficient. If you are having problems with users not being able to allocate enough memory, you may have to increase `NW_PGA_MAX_ALLOC`.

You may also have to adjust the Oracle initialization parameter, `PROCESSES`, to increase the number of Oracle connections. The `PROCESSES` parameter should reflect the maximum number of user connections you expect to have plus the number of Oracle background processes. You should also include some extra processes for administrative tasks.

Network

By making sure that you have a sufficient amount of packet-receive buffers, you avoid dynamically allocating them at run time. The packet-receive buffers are used to buffer the incoming network requests while the processor is busy. The number of packet-receive buffers is controlled by the NetWare parameter `MINIMUM PACKET RECEIVE BUFFERS`. This parameter is set in STARTUP.NCF. A good starting point is 200 (sufficiently high for about 40 users or fewer).

By monitoring the packet-receive buffers on the NetWare monitor screen, you can see whether they are being dynamically allocated. If the number of packet-receive buffers frequently exceeds the value you have set for `MINIMUM PACKET RECEIVE BUFFERS`, consider increasing this value. If you never achieve more than the minimum, you may be wasting space. By having the value too high, memory is wasted.

Try setting the value of `MINIMUM PACKET RECEIVE BUFFERS` down a little until you begin to see dynamic allocation at run time. At this point, increase the value of this parameter by 2 to 5 percent and monitor it on a regular basis. If dynamic allocation starts again, increase `MINIMUM PACKET RECEIVE BUFFERS` until that dynamic allocation stops.

SPX/IPX

To increase the number of connections that can be connected through SPX/IPX, increase the value of the Oracle parameter `SPX_MAX_CLIENTS`. Set this parameter to the maximum number of concurrent user connections required. If more users attempt to connect, they get an error message from SQL*Net.

If you get timeouts from the SPX listener threads, you may have to increase the value of the Oracle initialization parameter `SPX_LISTENERS`. An Oracle listener is responsible for spawning server processes on behalf of the user connection. SPX timeouts can occur when too many users try to connect at once.

TCP/IP

Under most circumstances, TCP/IP does not require any tuning. TCP/IP does not perform as well as SPX/IPX under NetWare but may be the best solution for you if other machines in your network are running TCP/IP. TCP/IP can be used for Oracle both independently or with SPX/IPX.

I/O Subsystem

As is true for all operating systems, it is very important to ensure that your performance is not bound by physical I/O rates. Be sure that random I/Os do not exceed the physical limitations of your disk drives. Refer to Chapters 14 and 15 for more details.

With NetWare, Asynchronous I/O (AIO) is always enabled. There is no need to adjust tuning parameters to ensure that AIO is enabled.

The default block size for Oracle on NetWare is 4096. This may be sufficient. If your data access patterns are primarily random, and the number of columns in a row is moderate to small, you may benefit by reducing the `DB_BLOCK_SIZE` to 2048. Doing so causes less data to be retrieved in each I/O and saves space in the SGA by having a smaller block buffer size.

If you have a mix of sequential and random data, and the row size is relatively large, you may be better off leaving the block size at 4096. Although the blocks take up more space in the SGA, the number of I/Os is significantly reduced. It doesn't take much more overhead and time to retrieve 4K of data from disk than it does to get 2K of data.

At the other extreme, if your data access is primarily sequential, you may benefit by setting the `DB_BLOCK_SIZE` to 8192. Because sequential access to the database reads the next block anyway, having a larger block size ensures that the database block is already in the SGA. If your data access is random, you end up using up unnecessary space in the SGA. The value you choose for the block size affects performance, either for the better or the worse. If you are unsure, leave the parameter at the default of 4096.

NetWare Summary

There are several ways you can improve the performance of Oracle on your NetWare system. NetWare is a robust, low-overhead operating system that may be the best solution for your RDBMS needs. With the new addition of NetWare SMP, you may be able to achieve highly scalable performance by adding additional CPUs.

The preceding sections have described various parameters you can adjust in the NetWare operating system and in Oracle itself; a few Oracle parameters are unique to NetWare. The most important area of tuning in NetWare is the amount of memory that can be dedicated to Oracle. You can maximize the memory available to Oracle by reducing the need for memory in other areas of NetWare. You also saw how to increase network performance by reducing dynamic allocation of buffers. Finally, you looked at how to maximize I/O usage by carefully choosing the correct block size.

Use these tuning guidelines in conjunction with other guidelines described in Chapters 9 and 10 to maximize the performance of the Oracle server. Tuning the server operating system in conjunction with the application and client results in the best-performing system possible.

Windows NT

Microsoft Windows NT is a relatively new OS that has quickly gained in popularity. Windows NT has file and print services similar to those offered by NetWare, but Windows NT is primarily used as an application server or client operating system. Windows NT comes in two varieties: NT Workstation and NT Server.

The NT Workstation product is designed for the client users and does not contain many of the management pieces that come standard with the NT Server product. The NT Server product includes the management tools necessary for maintaining a server.

Windows NT is a multiprocessor operating system and can take advantage of scaleable performance increases by adding additional CPUs.

As is NetWare, Windows NT is a server operating system. There is no facility within Windows NT to provide login functionality from terminals. From the very beginning, Windows NT has been designed to support clients over network connections. Unlike NetWare, Windows NT has much more functionality within the server itself.

The Windows NT operating system provides functionality such as 16-bit Windows application support and a GUI. Windows is very easy to manage using these graphical tools. But because of this additional functionality, Windows NT has greater overhead than NetWare.

Architectural Overview

Windows NT is based on a microkernel architecture. In a microkernel architecture, the core kernel is very small because most of the OS functionality is removed from the kernel. The kernel is the core of the operating system; it is where all base functionality exists. By removing most of the OS functionality from the kernel, the system is very modular; large pieces of the OS can be easily replaced.

Although a microkernel does not provide additional performance, it does provide a flexibility that OS and device driver developers can take advantage of. By having a flexible subsystem architecture, OS changes such as file systems, hardware architecture, and memory subsystems can easily be replaced.

In Windows NT, hardware support is provided through the Hardware Abstraction Layer (HAL). The HAL is provided by hardware vendors to support the base architecture of their system as well as to provide device driver support. Because of the microkernel and HAL, Windows NT can support different architectures such as Intel, DEC Alpha, MIPS, PowerPC, and so on. The microkernel is the common code base on which each of these architectures is based.

Most 16-bit applications written for Windows can run on non-Intel architectures by using a compatibility mode driver. Applications written specifically for Windows NT can be ported to non-Intel platforms with minimal effort. However, applications that have not been written to take advantage of a particular architecture may not run optimally.

Oracle is currently supported only on the Intel architecture. As with all Oracle ports, Oracle has been optimized for Windows NT and specifically optimized for Windows NT on Intel platforms.

The Windows NT architecture provides for the use of *threads*, sometimes known as *lightweight processes*. By using threads instead of processes, much of the overhead associated with process switching is reduced. Threads are automatically created when the Oracle instance is started (see Table 12.1).

Table 12.1 Oracle Service Threads

<i>Thread Number</i>	<i>Oracle Process</i>
0, 1	Oracle Service
2	PMON
3	DBWR
4	LGWR
5	SMON
6	RECO

A shadow thread is created on behalf of each user accessing the Oracle database. Remember that the shadow processes communicate with the user and interact with Oracle to carry out the user's requests. For example, if the user process requests a piece of data not already in the SGA, the shadow process is responsible for reading the data blocks from the data files into the SGA. Under Windows NT, these processes are invoked as kernel threads. These threads each have their own thread number.

Tuning Considerations

With Windows NT, probably the most important tuning consideration is ensuring that sufficient physical memory is available to run Oracle and the required number of server processes. Windows NT uses virtual memory, which means that Oracle and user processes can allocate an almost unlimited amount of memory through paging. If you are not careful, you may overconfigure the amount of virtual memory you are using and exceed the amount of physical memory in the system. If this occurs, the system begins paging and performance is severely degraded.

In addition to the concern about memory, other areas of importance in tuning the Windows NT system include the network and I/O subsystems and the reduction in OS overhead.

Memory

Windows NT is the opposite extreme from the NetWare system. Where NetWare uses no virtual memory, Windows NT uses only virtual memory. Because all memory in the system is treated equally, without limitations, it is important to be careful of user processes consuming large amounts of memory and paging out the SGA. In Windows NT, there are no limitations on memory used for user processes, disk caching, print caching, and so on; thus, it is best to dedicate the Windows NT server to either file and print services or to application services.

If you use Asynchronous I/O (AIO) and Oracle can lock down memory for AIO, the database block buffers are not swappable. Monitor the system on a regular basis to make sure that no paging is occurring in the system.

Reduce Unnecessary Memory Usage

You can take some measures to reduce the amount of memory used by Windows NT. Use the Control Panel's Network Settings screen to choose the Maximize Throughput for Network Applications option. This optimizes the server memory for network applications, reducing some of the file system caching and overhead in memory management. Also remove any network protocols not needed by the system to cut down on system overhead and memory usage through the Control Panel.

Also use the Control Panel to turn off any services you are not using. Doing so reduces memory usage and CPU overhead. By reducing all the unnecessary services, you can increase the performance of the system.

SGA Tuning

To maximize performance, allocate as much memory as possible to the SGA. Use the techniques discussed in Chapters 9 and 10 to determine whether the memory can best be used for the shared pool or for database block buffers.

Because the SGA resides in virtual memory, it is important that you not allocate so much memory for the SGA that the system pages. The overhead of paging and swapping overshadows any benefit you may receive from a larger SGA.

Remember to save memory for the user processes. You should frequently monitor your system to make sure that no paging occurs at any time.

User Capacity

You can easily determine the amount of memory necessary for your application on a per-user basis. Start up Oracle and note the amount of memory available by using the Windows NT performance monitor. Monitor the Available Bytes option under Memory in the performance monitor. Once users begin accessing the application in a typical manner, record the amount of memory again. Take the difference and divide this result by the number of users accessing the application. Multiply this per-user memory usage value by the maximum number of users who may be connected to the application to determine the amount of memory you must reserve for user connections. Be sure to add a little extra memory just in case.

Unlike NetWare, the size of a user's PGA is not bound by any initialization parameters. Because of this, be careful that a user's PGA does not consume too much system memory.

To increase the number of Oracle connections, you may also have to adjust the Oracle initialization parameter `PROCESSES`. The `PROCESSES` parameter should reflect the maximum number of user connections you expect to have plus the Oracle background processes. You should also include some extra processes for administrative tasks.

Network

Little network tuning is necessary with Windows NT. However, performance can be enhanced by prioritizing the network bindings. You can prioritize the network bindings with the Control Panel's Network Configuration utility. By putting the network protocol you use most frequently first in the list, that protocol gets highest priority. Place other network protocols in the list in the order in which you use them.

Removing any protocols you do not use can also benefit performance by both reducing memory consumption and CPU overhead. Also be sure that the system is configured as a *server* in the network configuration screen.

I/O Subsystem

As is true for all operating systems, it is very important to ensure that performance is not bound by physical I/O rates. Be sure that random I/Os do not exceed the physical limitations of the disk drives. Refer to Chapters 14 and 15 for more details.

With Windows NT, Asynchronous I/O (AIO) is always enabled. There is no need to adjust any tuning parameters to ensure that AIO is enabled.

The default block size for Oracle on Windows NT is 2048. This may be sufficient. If you have a very large database or the data access is primarily sequential, you may want to increase `DB_BLOCK_SIZE` to 4096. Although the data blocks take up more space in the SGA, the number of I/Os done is significantly reduced. It doesn't take very much more overhead and time to retrieve 4K of data from the disks than it does to get 2K of data.

If data access is primarily sequential, you may benefit by setting the `DB_BLOCK_SIZE` parameter to 8192. Because sequential access to the database will read the next block anyway, larger block sizes read that data into the SGA before you need it. If your data access is random, you end up wasting space in the SGA. The value you choose for the block size affects performance, either for the better or the worse. If you are unsure, leave the parameter at the default of 2048.

When creating Windows NT file systems, you have several choices.

- ◆ FAT
- ◆ HPFS
- ◆ NTFS

Although each of these performs well in certain situations, I recommend using NTFS. NTFS provides you with the best level of overall performance and is the file system that Microsoft endorses.

Windows NT Summary

There are several ways you can improve the performance of Oracle on your Windows NT system. The primary areas of concern are reducing OS overhead and making sure that the system is not paging. As with NetWare SMP and UNIX, you can get scalable performance by adding additional CPUs.

The preceding sections have described various parameters you can adjust in the Windows NT operating system as well as in Oracle itself. The most important area of tuning in Windows NT is the amount of memory that can be dedicated to Oracle. You learned of several ways to maximize the memory available to Oracle by reducing some of NT's overhead. You also learned how to maximize I/O usage by carefully choosing the correct block size.

Use these tuning guidelines in conjunction with other guidelines described in Chapters 9 and 10 to maximize the performance of the Oracle server. Tuning the server operating system in conjunction with the application and client results in the best-performing system possible.

OS/2

Like NetWare, OS/2 is an operating system that has been around for some time. OS/2 has just recently added support for SMP computers with the release of OS/2 for Symmetrical Multi-processing version 2.11. This new version of OS/2 provides the additional performance necessary to become a viable server operating system.

Like Windows NT, OS/2 is primarily designed for the mass market as both a desktop and server operating system; it provides a graphical user interface and does not require very much user intervention for system tuning and configuration.

Architectural Overview

OS/2 is a layered architecture. At the center of the OS/2 is the core, or kernel, of the OS; the device drivers and OS subsystems are on the outer layers of the OS. OS/2 is a virtual memory operating system that uses an OS file for the paging file. The OS/2 operating system includes virtual DOS machines; like Windows NT, OS/2 can run most DOS and Windows programs. OS/2 does not have many tuning parameters, and those parameters that can be changed are typically in the CONFIG.SYS file.

Tuning Considerations

As is true for most operating systems, probably the most important tuning consideration is ensuring that there is sufficient physical memory available to run Oracle and the required number of server processes. OS/2 uses virtual memory, which means that Oracle and user processes can use an unlimited amount of memory through paging. If you are not careful, you may overconfigure the amount of virtual memory you are using and exceed the amount of physical memory in the system.

In addition to the concern about memory, other areas of importance in tuning the OS/2 system include the network and I/O subsystems and the reduction in OS overhead.

Memory

Like Windows NT and UNIX, OS/2 is a virtual memory system, which means that you can allocate memory and run programs that use more memory than physically exists on the system. Although this works and operates very well, doing so is undesirable in a RDBMS environment. Because the SGA and server processes are very efficient, any paging or swapping that occurs severely degrades performance. As with all virtual memory operating systems, you should avoid paging and swapping.

Reduce Unnecessary Memory Usage

Because memory is usually a critical resource, anything you can do to reduce the amount of memory allocated to non-Oracle processes helps free memory that can be used for additional SGA memory or server processes. One way to reduce unnecessary memory is to reduce the size of the disk cache.

Because Oracle provides for its own disk cache (in the form of the SGA), it is unnecessary to provide large amounts of memory for OS disk caching. You can reduce the amount of OS disk caching by adjusting the value of the cache parameter for the IFS line in CONFIG.SYS for HPFS or HPFS386. If you are using FAT, there is a separate line, DISKCACHE. This value should be reduced, but probably to less than 2048. A reasonable value for both these values is 2048.

SGA Tuning

To maximize performance, allocate as much memory as possible to the SGA. Use the techniques discussed in Chapters 9 and 10 to determine whether the memory can best be used for the shared pool or for database block buffers.

Although it is possible in a virtual memory system to allocate more memory for the SGA than there is physical memory, you should avoid this practice. The overhead of paging and swapping outweighs any benefits you may see from a larger SGA.

Remember that user processes also require large amounts of memory. You should frequently monitor your system to make sure that no paging is occurring at any time.

User Capacity

The method used to determine the per-user memory requirements for OS/2 is similar to the method used for Windows NT: Start up Oracle and record the amount of available memory by using the OS/2 performance monitor or the SMP performance monitor. Record the amount of memory being used. Once a specified number of users begin to access the application, record the amount of memory again. Take the difference between these values and divide the result by the number of users accessing the application. This value is the per-user memory usage. Multiply this value by the maximum number of users who may connect to the application to determine the amount of memory you must reserve for user connections. Be sure to add a little extra memory just in case.

Unlike NetWare, the size of a user's PGA in OS/2 is not bound by any initialization parameters. Because of this, be careful that a user's PGA does not consume too much system memory.

To increase the number of Oracle connections, you can adjust the Oracle initialization parameter PROCESSES. The PROCESSES parameter should reflect the maximum number of user connections you expect to have plus the Oracle background processes. You should also include some extra processes for administrative tasks.

Network

With OS/2 as with Windows NT, it is unnecessary to adjust any networking parameters. The OS/2 system is designed to be accessed through a network and does not require any tuning.

I/O Subsystem

As is true for all operating systems, it is very important to ensure that performance is not bound by physical I/O rates. Be sure that random I/Os are not exceeding the physical limitations of the disk drives. Refer to Chapters 14 and 15 for more details.

With OS/2, as with Windows NT and NetWare, Asynchronous I/O (AIO) is always enabled. There is no need to adjust any tuning parameters to ensure that AIO is enabled.

The default block size for Oracle on OS/2 is 2048. This may be sufficient. If you have a very large database or the data access is primarily sequential, you may want to increase `DB_BUFFER_SIZE` to 4096. Although the blocks take up more space in the SGA, the number of I/Os is significantly reduced. It doesn't take very much more overhead and time to retrieve 4K of data from the disk than it does to get 2K of data.

If your data access is primarily sequential, you may benefit by setting `DB_BLOCK_SIZE` to 8192. Because sequential accesses to data definition uses the next rows in the database, when you read more of those rows with a larger block size, the rows are already in the SGA when you are ready to use them. This prefetching cuts down on I/O overhead. If you are accessing the database in a random fashion, prefetching ends up wasting space in the SGA. The value you choose for the block size affects performance, either for the better or the worse. If you are unsure, leave the parameter at the default of 2048.

When you are creating OS/2 file systems, you have three main choices:

- ◆ FAT
- ◆ HPFS, sometimes known as HPFS286
- ◆ HPFS386

Of the these three choices for file systems, I generally recommend HPFS386. Although there are certain conditions under which FAT performs better, in general, you will see the best performance from HPFS386. Under most situations, especially under Oracle, HPFS386 provides the best performance and flexibility

OS/2 Summary

OS/2, like Windows NT, is designed to run applications out of the box with little or no system tuning. As such, most of the areas of concern in an OS/2 environment focus around resource allocation rather than on tuning. The primary areas of concern are reducing OS overhead and making sure that your system is not paging. With

OS/2, as with NetWare SMP, Windows NT, and UNIX, you can get scaleable performance by adding additional CPUs.

The preceding sections described a few parameters you can adjust in the OS/2 operating system as well as in Oracle itself. The most important area of tuning in OS/2 is the amount of memory that can be dedicated to Oracle. You learned about several ways to maximize the memory available to Oracle by reducing some of OS/2's overhead. You also learned about some slight performance gains you can obtain by carefully choosing your file system. Finally, you learned how to maximize I/O usage by carefully choosing the correct block size.

Use these tuning guidelines in conjunction with other guidelines described in this book to maximize the performance of the Oracle server. Tuning the server operating system in conjunction with the application and client provides for the best-performing system possible.

UNIX

The UNIX operating system has been around since 1969. It predates NetWare, Windows NT, and OS/2 by many years. The UNIX operating system is different from NetWare, Windows NT, and OS/2 in that it was not designed as a client/server operating system. That is not to say that UNIX is not now used as a network server, but that network service was not the original intent of the operating system.

Even though UNIX is fairly standard, there has been a divergence in UNIX over the years. Because almost every major computer company has a version of UNIX that they develop and sell, there are differences in the UNIX offerings from the different vendors.

In the PC server market, there are three main UNIX versions: SCO UNIX, SCO UnixWare, and Solaris from SunSoft. IBM calls their UNIX offering AIX. HP has a UNIX operating system called HP-UX; AT&T Global Information Solutions simply calls their product UNIX System V. SunSoft produces versions of UNIX for both Intel and Sun SPARC processors; their products are called Solaris and Solaris X86.

Over the years, UNIX has evolved. Currently, most vendors (with the exception of SCO) base their versions of UNIX on one common core: UNIX System V Release 4, which is sometimes referred to as UNIX SVR4.

In many cases, applications are binarily compatible between OSes on the same platform. But it is not uncommon for the vendor to put special enhancements into the operating system for performance. Oracle always takes advantage of OS-specific features, even though it would be simpler to ship just one binary.

Architectural Overview

The UNIX operating system consists of a core piece called the *kernel* surrounded by applications and tools. The kernel contains all hardware support, device drivers, scheduling routines, and the network stack. Unlike the microkernel architecture used in Windows NT, the UNIX kernel contains all the core OS functionality.

UNIX is a virtual memory operating system but is very configurable. In the UNIX OS, not all memory is allocated and used in the same manner. Shared memory used for the Oracle SGA is treated differently from normal memory. Shared memory is allocated at boot time and is not available to general user processes. Shared memory must be allocated through shared memory system calls in the operating system.

The fact that shared memory is treated differently allows certain functions to occur. Some varieties of UNIX allow you to allocate shared memory using a 4M memory page size. This arrangement cuts down on page entries that must be maintained by the operating system and guarantees larger chunks of contiguous memory. Shared memory is also unique in that it is non-pageable. This means that if you have enough shared memory to create the SGA at instance startup, you don't ever have to worry about the SGA being paged out.

Over the years, other features that have been added to the UNIX operating system to improve database performance include real-time processing, high-speed semaphores, and Asynchronous I/O. Not all these features are in every implementation of UNIX. Check with your particular OS vendor to see what is available for your environment.

Tuning Considerations

As is true for all the operating systems described in this chapter, the most important tuning consideration is ensuring that sufficient memory is available to run Oracle and the required number of server processes. Because UNIX is a virtual memory operating system, you can always start more server processes, but if you are not careful, you may begin paging.

Many varieties of UNIX have extended features such as the post-wait semaphore and Asynchronous I/O available to enhance Oracle performance. Other areas of importance in tuning the UNIX system include both the network and I/O subsystem.

Memory

Memory is very important in the UNIX operating system. As you have seen in earlier chapters, the way to increase performance is to maximize the use of the SGA to avoid costly disk I/Os. In UNIX as in all OSes, it is important to avoid paging. Because the SGA is in shared memory in UNIX, the SGA is guaranteed not to page—but the server processes can page if too much memory is being used.

It is important to allocate enough shared memory to accommodate the SGA. If you do not have enough memory to create the SGA, the instance does not start. If this happens, you must either configure the OS to allow more shared memory or reduce the size of the SGA. If you are using enhanced 4M shared memory pages, you can allocate the amount of shared memory only in 4M units.

Be careful that you do not allocate so much shared memory for the SGA that user processes and server processes page. Be sure to monitor the system periodically to make sure that no paging is occurring. To maximize memory used by the SGA, reduce all unnecessary memory used by the operating system and limit the amount of memory that can be allocated by users.

Reduce Unnecessary Memory Usage

One of the best ways to free up memory for Oracle is to free up some memory used by the file system buffers. The file system buffers are used by the operating system to cache data. As you know, there can be a significant performance increase when accessing files that have been cached. But Oracle uses a feature called *synchronous I/O* to ensure that writes to the disk are not returned until the data has actually been written. Because Oracle must guarantee that I/Os have been written to the disk to ensure data integrity, OS disk write caching cannot be done.

By default, the number of file system buffers is determined by the amount of memory in the system. Because Oracle bypasses the disk cache on writes and uses the SGA for reads, you really don't need a large number of file system buffers. By reducing this number, you may see a slight performance decrease with OS operations but any additional memory allocated to the SGA increases Oracle performance.

To tune the OS buffer cache, use the following guidelines:

UNIX Version	Parameters To Set for OS Buffer Cache
SCO UNIX	The total number of buffers allocated to the disk cache is determined by the value of <code>NBUF</code> in the STUNE file.
UnixWare	The number of buffers allocated to the disk cache is dynamically allocated in <code>NBUF</code> chunks to a maximum of <code>BUFHWM</code> in the STUNE file.
Solaris	The number of buffers allocated to the disk cache is controlled by <code>BUFHWM</code> in <code>/ETC/SYSTEM</code> .

By reducing the unnecessary memory used for disk caching, more memory can be allocated to the SGA. Don't reduce the disk cache so far that it is difficult to run OS commands and access Oracle parameter files. Do not set the disk cache buffers less than 600. These values represent the number of 512 byte blocks.

SGA Tuning

To maximize performance, allocate as much memory as possible to the SGA. Use the techniques discussed in Chapters 9 and 10 to determine whether memory is best used for the shared pool or for database block buffers.

In UNIX, the shared memory area used by Oracle for the SGA is usually contiguous. However, if you have multiple instances of Oracle that have started and stopped several times, the shared memory area may no longer be contiguous. If you use 4M pages, you are guaranteed to have at least 4M of contiguous memory.

The amount of memory allocated for shared memory is the product of two OS-tunable parameters: SHMMAX and SHMSEG. SHMMAX specifies the maximum size of a shared memory segment; SHMSEG specifies the maximum number of shared memory segments available in the system. Applications are responsible for allocating only the amount of shared memory they need and so do not waste space.

In SCO UNIX and UnixWare, the parameters SHMMAX and SHMSEG are located in the UNIX parameter file /ETC/CONF/CF.D/STUNE. In Solaris, the shared memory parameters are set in the file /ETC/SYSTEM. Your OS administrator's manual should have more information on setting system parameters.

NOTE: It is much more efficient for Oracle to have one large shared memory segment than several smaller ones. Therefore, set SHMMAX to have a value larger than the size of the SGA.

To allocate 4M pages for shared memory, use the following parameters in the STUNE file:

UNIX Version	Parameters To Set for Shared Memory
UnixWare	With UnixWare, the parameter PSE_PHYSMEM must be set to the total amount of shared memory on which you want to use 4M pages. This value is rounded up to the nearest multiple of 4 megabytes.
Solaris	With Solaris, the maximum amount of 4M pages is a percentage of your total system memory. See your Solaris documentation for details.

NOTE: In UNIX implementations that use 4M pages, there is usually a threshold at which 4M pages are used. This threshold is approximately 3M. If you allocate a 2.5M shared memory segment, you do not get a 4M page.

The amount of memory allocated to a single user must also be tuned in the UNIX operating system. Because Oracle is treated the same as any other user, you must allocate enough memory for Oracle to use for the SGA and the server processes. The amount of memory available for users is tuned with the following parameters:

<i>UNIX Version</i>	<i>Parameters To Set for User Processes in Memory</i>
SCO UNIX	The total amount of memory that can be used by a user process is set by the parameter <code>MAXUMEM</code> in the <code>/ETC/CONF/CF.D/STUNE</code> file.
UnixWare	The total amount of memory that can be used by a user process is determined by the OS parameters <code>SVMMLIM</code> and <code>HVMMLIM</code> in the <code>/ETC/CONF/CF.D/STUNE</code> file.
Solaris	The total amount of memory that can be used by a user process is determined by the OS parameter <code>HVMMLIM</code> in the <code>/ETC/SYSTEM</code> file.

Make sure that sufficient memory is available so that Oracle can allocate for the SGA. Remember to save memory for the user processes as well. You should frequently monitor your system to make sure that no paging is occurring at any time.

User Capacity

You can easily determine the amount of memory necessary for your application on a per-user basis. Start up Oracle and note the amount of available memory with the UNIX utility `sar -r`. The output from `sar -r` consists of `freemem` (Free Memory Pages) and `freeswp` (Free Swap Pages). The value given in the `freemem` column is the number of 4K pages available. Once users begin accessing the application in a typical manner, record the amount of memory again. Take the difference and divide this result by the number of users accessing the application. This value is the per-user memory usage. Multiply this value by the maximum number of users who may connect to the application to determine the amount of memory you must reserve for user connections. Be sure to leave a little extra memory just in case.

The size of a user's PGA is not bound by any initialization parameters. Because of this, be careful that a user's PGA does not consume too much system memory.

The UNIX operating system parameters `MAXUP` and `NPROC` must also be set to allow a sufficient number of users to connect. Remember that, when users connect with Oracle, an Oracle shadow process is created. The shadow process is created under the Oracle user ID. Therefore, you must increase not only the number of processes system wide, but also the per-user process limits.

The per-user process limit is set with the OS parameter **MAXUP**. The maximum number of processes system wide is set by the OS parameter **NPROC**. Both of these values are in the STUNE file. **NPROC** should be at least 50 greater than **MAXUP** to account for OS processes.

To increase the number of Oracle connections, you may also have to adjust the Oracle initialization parameter **PROCESSES**. The **PROCESSES** parameter should reflect the maximum number of user connections you expect to have plus the Oracle background processes. You should also include some extra processes for administrative tasks.

Network

With UNIX, the amount of network tuning is usually very minimal. Because the amount and type of tuning necessary is implementation dependent, the following sections look at each implementation separately.

SCO UNIX

SCO UNIX has support for TCP/IP, SPX/IPX, and Banyan Vines; it also supports both SQL*Net 1 and SQL*Net 2. You can have one or more of these protocols running at the same time.

With large numbers of connections, it may be necessary to increase the number of sockets available in the system. To increase the number of sockets, use the SCO UNIX utility **netconfig**.

You may also have to increase the number of pseudo ttys. Pseudo ttys are used to provide user **telnet** and **rlogin** sessions. If you run your applications in a client/server manner, increasing the number of pseudo ttys is not necessary. If you run your application through login sessions, you may have to increase the number of pseudo ttys available in the system. You can increase the number of pseudo ttys through the SCO UNIX utility **netconfig**.

SCO UNIX networking is based on **streams**. The streams design allows for various streams buffers to be passed from the device driver up through several layers of streams modules. These modules allow different protocols to be easily incorporated into the network subsystem. There are various sizes of streams buffers in the system.

If a streams buffer is not available, the next largest size is allocated. This dynamic reallocation of streams buffers is inefficient. Using **crash** or some other utility, determine whether there are any failures on streams buffers of a certain size. If you see failures, increase the number of buffers of that size. My rule of thumb is to keep doubling the number of buffers until there are no more failures.

NOTE: In SCO UNIX version 5, streams buffers grow dynamically. To avoid the inefficient dynamic growth, monitor the number of streams in use during peak times and set the default values slightly larger than peak values.

By keeping the number of streams buffer allocation failures to a minimum, you can achieve maximum network efficiency.

To support a sufficient number of **SPX/IPX** connections under SCO UNIX, you must increase the value of the parameters `SPX_MAX_SOCKETS` and `IPX_MAX_SOCKETS` in the OS file `/etc/conf/pack.d/ipx/ipx_tune.h`. You may also have to tune the value `SPX_MAX_CONNECTIONS` in the OS file `/etc/conf/pack.d/spx/spx_tune.h`. The value of `IPX_MAX_SOCKETS` should be twice that of `SPX_MAX_SOCKETS`. These values should reflect the maximum number of users you expect to be connected through SPX/IPX.

Under normal loads **TCP/IP** need not be tuned. However, if you seem to be losing connections under extreme loads, increase the value of `TCPWINDOW` in the file `/etc/conf/pack.d/tcp/space.c`. Try setting the value to 24576 or higher.

UnixWare

UnixWare supports TCP/IP and SPX/IPX as well as both SQL*Net 1 and SQL*Net 2. You can have one or more of these protocols running at the same time. Typically, there are no more parameters that must be tuned for networking.

The number of **SPX/IPX** connections need not be tuned.

As with SCO UNIX, under normal loads with UnixWare, **TCP/IP** need not be tuned. However, if you seem to be losing connections under extreme loads, increase the value of `TCPWINDOW` in the file `/etc/conf/pack.d/tcp/space.c`. Try setting the value to 24576 or higher.

To support a large number of connections (over 512) with TCP/IP, it is necessary to modify the `/etc/conf/sdevice.d/tcp` file. The third column in this file specifies the number of sockets to create. You may have to increase this number to support the required number of users.

Solaris

Solaris networking is very dynamic; you don't have to tune any parameters. Solaris supports many connections without administrative intervention.

SPX/IPX is available as an add-on package for Solaris x86 and does not require any additional tuning.

TCP/IP under Solaris does not require any tuning.

I/O Subsystem

As with all other operating systems described in this chapter, it is important to ensure that UNIX performance is not bound by physical I/O rates. Be sure that random I/Os do not exceed the physical limitations of the disk drives. Refer to Chapters 14 and 15 for details.

With UNIX, you have the choice of using the UNIX file system for your data storage or the raw device interface. This choice is not always an easy one to make. The raw device interface is more difficult to manage but provides a higher level of performance. File system files are much easier to use but have more overhead associated with them.

File System

Using the UNIX file system is easier than using raw devices. Using file system files, Oracle simply creates the file. However, when using the file system, Oracle must contend with the UNIX disk caching system and use synchronous writes to ensure that the write request does not return to the DBWR or LGWR before it has actually written the data to disk.

With the UNIX file system, there is also the additional overhead of the data being read into the UNIX disk cache and then being copied to the SGA. This arrangement causes additional overhead on reads.

Finally, when you use the file system, you are not guaranteed to have contiguous blocks on the disk—in fact, you are almost guaranteed not to have contiguous blocks.

Raw Device Interface

The raw device interface allows for the least amount of overhead you can achieve with UNIX I/Os. When using the raw device interface, UNIX simply assigns a section of the disk to each raw device. This portion of the disk is contiguous; accesses to it bypass all disk caching and file system overhead.

Raw devices are not as easy to manage because the device is considered one big chunk of data for the operating system. Backup and recovery must be handled slightly differently because file copies do not work, and the size of the raw device cannot be changed once it is created. Backup operations must be done using the UNIX `dd` command or by a third-party backup utility that supports raw devices.

Raw devices give greater performance with less overhead and are fully supported by Oracle. Whether or not you use raw devices is a decision you must make based on its ease-of-use and increased performance.

Asynchronous I/O

With UNIX, AIO is not always enabled. It is necessary to enable AIO in both the OS and in Oracle. By using AIO, the DBWR can manage many I/Os at once, eliminating the need for multiple DBWR processes. List I/O allows the DBWR to pass to the OS a list of AIO commands, reducing the number of calls it must make.

For **SCO UNIX**, the following OS parameters for Asynchronous I/O should be set in the /ETC/CONF/CF.D/STUNE file:

<i>Parameter</i>	<i>Comments</i>
NAIOREQ	This parameter represents the maximum number of pending AIO requests that can occur in the system. Set it to 512.
NAIOBUF	This parameter specifies the size of the AIO buffer table. Set it to 512.
NAIOREQPP	This parameter represents the maximum number of pending AIO requests on a per-user basis. Set it to 512.

For **SCO UNIX**, the following Oracle initialization parameter for Asynchronous I/O should also be set:

<i>Parameter</i>	<i>Comments</i>
ASYNC_WRITE	This parameter tells Oracle that the DBWR should use Asynchronous I/O. If AIO is available, set this parameter to TRUE.

NOTE: With SCO UNIX, synchronous I/O is available only if you are using the raw device interface. If your data files reside on the UNIX file system, AIO is not available to you. If you cannot use AIO, you can compensate by adding DBWR processes. You should have one or two DBWR processes per data disk. Use the parameter `DB_WRITERS` to increase the number of DBWR processes.

For **UnixWare**, the following OS parameters for Asynchronous I/O should be set in the STUNE file:

<i>Parameter</i>	<i>Comments</i>
NUMAIO	This parameter represents the maximum number of AIO control blocks. It essentially specifies the maximum number of AIOs in the system. Set it to 512.
AIO_LISTIO_MAX	This parameter specifies the size of <code>listio</code> requests. <code>listio</code> allows a list of AIO requests to be specified. Set this parameter to 512.

For **UnixWare**, the following Oracle initialization parameters for Asynchronous I/O should also be set:

<i>Parameter</i>	<i>Comments</i>
USE_ASYNC_IO	This parameter tells Oracle that the DBWR should use Asynchronous I/O. Set it to TRUE.
LGWR_USE_ASYNC_IO	This parameter tells Oracle that the LGWR should use Asynchronous I/O. Set it to TRUE.

NOTE: In UnixWare, Asynchronous I/O is available whether you are using the file system or the raw device interface. It is very important to make sure that the device /DEV/ASYNC is set to read-write for the Oracle user account or that you change the ownership of that device to Oracle. If this change is not made, Oracle is unable to use Asynchronous I/O.

For **Solaris**, you do not have to set any OS parameters to configure Asynchronous I/O. Solaris can support large numbers of outstanding AIO requests and is not tunable.

However, to initialize Oracle to take advantage of AIO, the following Oracle initialization parameters should be set:

<i>Parameter</i>	<i>Comments</i>
USE_ASYNC_IO	This parameter tells Oracle that the DBWR should use Asynchronous I/O. Set it to TRUE.
LGWR_USE_ASYNC_IO	This parameter tells Oracle that the LGWR should use Asynchronous I/O. Set it to TRUE.

NOTE: In Solaris, Asynchronous I/O is available whether you are using either the file system or the raw device interface.

You should always use Asynchronous I/O (if possible). When you use Asynchronous I/O, you can keep the number of DBWR processes at 1 and therefore reduce process overhead.

Miscellaneous

Over the years, many features have been added to the different versions of UNIX to improve Oracle performance as a result of performance tuning and testing. Many of these features have been the result of benchmarking efforts. Benchmarking is probably the best way to accurately measure the performance of the system and any gains that occur when new features are added. Although it is unusual for an RDBMS vendor to put a feature into their product solely for the purpose of improving performance on a single benchmark, it has happened. However, since the TPC put a clause into each benchmark specification prohibiting “benchmark-special” features, this has not occurred.

The best way to judge the effect of a new feature is in a controlled environment. By using a benchmark or a custom test you have developed, you can quantifiably measure performance gains. Not only are features added to Oracle but to the OS as well. The following sections list some of these features.

Post-Wait Semaphore

The post-wait semaphore is designed to allow Oracle to have more control over the scheduling of processes (refer to Chapter 11, “Tuning the Server Operating System”). This new type of semaphore allows Oracle to signal sleeping processes and start them working again, significantly reducing idle CPU cycles. The post-wait semaphore is available in most varieties of the UNIX operating system except for Solaris. In Solaris, a different scheduling algorithm and high-speed semaphores have made the post-wait semaphore unnecessary.

The post-wait semaphore was developed by Oracle and several UNIX operating system vendors. Post-wait consists of a device driver and a post-wait device. Access to the post-wait device is through an `iocntl` call into that driver. When the post-wait driver is enabled, both within the OS and within Oracle, the use of traditional semaphores and sleep or alarm calls is eliminated.

To enable post-wait within Oracle, set the initialization parameter `USE_POST_WAIT_DRIVER` to `TRUE`; set the initialization parameter `POST_WAIT_DEVICE` to the name of the post-wait device. The default initialization file provided with Oracle should have the default post-wait device name (typically `/dev/pw`). See your installation and tuning guide for details.

Post-wait should offer immediate and fairly significant performance improvements. No risk or performance degradation can occur from using the post-wait semaphore. I recommend that you always use post-wait when it is available.

Intimate Shared Memory

Intimate Shared Memory (ISM) is available on the Solaris operating system. ISM allows for sharing of page tables for shared memory. This feature is also used as the vehicle to enable the 4M pages described earlier in this chapter. ISM can also lock pages into memory, reducing the overhead for doing I/O. Oracle enables this feature when the initialization parameter `USE_ISM` is set to `TRUE`.

When it is available, you should turn on ISM to enable the 4M pages and reduce overhead. By locking the shared memory region into memory and making 4M contiguous pages available, overhead is reduced and performance is enhanced.

Scheduler

Several scheduling options can enhance Oracle’s performance on the UNIX operating system (refer to Chapter 11). These options include the disabling of preemptive scheduling, load balancing, and cache affinity.

By disabling **preemptive scheduling**, you can take advantage of the fact that typical Oracle processes are of short duration. When you preempt short-running processes, you can waste CPU cycles performing the task switch when, in fact, the running process most likely would have relinquished the CPU in a short time anyway. With preemptive scheduling, you can

allow processes to run to completion. In this case, *completion* usually means that the process does an I/O operation, relinquishes the CPU, and goes to sleep waiting for the I/O to complete.

Preemptive scheduling is disabled in SCO UNIX by setting both the `preemptive` and `load_balance` variables to zero in the file `/etc/conf/pack.d/crlry/space.c`. These two parameters are similar. When `preemptive = 1`, a process on the same processor can preempt a lower-priority process on that CPU. When `load_balance = 1`, a process on one CPU can preempt a process on another CPU. By disabling these variables, processes run to completion.

CAUTION: When you disable preemptive scheduling, there is a danger that a runaway process may run until its maximum time slice has been used. By default, this is set to 1 second. You can avoid this by setting the UNIX parameter `MAXSLICE` to 3 or 4.

`MAXSLICE` represents the maximum number of ticks a process can run before it must relinquish the CPU. A *tick* in UNIX is typically 10 milliseconds. If you have very fast processors, you may be able to set this parameter to 1 or 2. Even when `MAXSLICE` is set to 1 or 2, it is unlikely that a process will hit that limit.

By disabling **cache affinity**, you may be able to reduce some unnecessary overhead. Cache affinity is used to attempt to run a process on the last processor it ran on. The theory behind this feature is that if any data is left in the CPU cache from the last time the process ran, the process will be able to take advantage of it. Running Oracle this way may or may not be a good thing.

If you have a heavily loaded machine with a large number of users (such as in a large OLTP system), it is extremely unlikely that there will be any data left in the CPU cache when it is time for your process to run again. In effect, you have just wasted many CPU cycles in the OS scheduler trying to relocate a process for no good reason.

On the other hand, if you are running a large application with a small number of processes (such as a DSS system), you may benefit from cache affinity. If you have a fair number of processes all running the same shared code, you will also see a benefit from cache affinity.

By carefully analyzing your workload, you should come to a logical conclusion about whether cache affinity will be a benefit for you. If you are not sure, it is probably best to leave the system in its default configuration.

UNIX Summary

Over the years, many features have been added to the various implementations of UNIX to improve the performance of Oracle. Part of the reason that these improvements have occurred is that many hardware vendors have a proprietary UNIX implementation and these vendors have worked closely with Oracle to improve RDBMS performance. Because UNIX has been around and stable for many years, there has been ample opportunity to test and analyze system performance.

As you have noticed, there is much more tuning and configuration involved when using the UNIX operating system than there is when you use NetWare or Windows NT. When choosing an OS, you must weigh the importance of configurability and control versus ease of use.

Recently, SCO UNIX merged with Novell UnixWare and Hewlett Packard. As a result of this merger, a new operating system will evolve in the next few years. This new operating system will have some of the characteristics of SCO UNIX and UnixWare. All the tuning concepts discussed here will still apply; however, the specific parameters may differ slightly.

Summary

This chapter looked at several major operating systems in detail. It is important to realize that this data is volatile and may have changed slightly by the time you read this. The database and OS marketplace is changing quickly.

Windows NT is a relatively young OS—and NetWare SMP and OS/2 are even younger. As these and other OSes evolve, the methods of tuning and configuring these OSes is also changing. This chapter attempted to provide the methodology and function of the tuning changes as well as the parameters themselves. It is unlikely that the functionality of these parameters will change. However, it is likely that the things which typically require intervention will become more dynamic; already, the interfaces to these parameters are now done with graphical interfaces.

Always refer to the documentation that comes with your operating system to check for changes to the parameters listed in this chapter. As operating systems grow and are modified to support new devices and architectures, the performance characteristics may change. You should always do careful analysis and testing before putting any tuning changes into production.

Chapter 3

System Processors

This chapter gives a brief overview of how system processors work and how performance is affected by various aspects of the computer hardware architecture. Of course, you cannot “tune” your CPUs, but you should look at how the hardware works to better understand the system.

This chapter first looks at the various components of computer hardware and then looks at some of these components in more detail. It is important to remember how the components react with each other to understand why you favor certain components such as memory.

Overview of Computer Architecture

Your computer system is made up of hundreds and thousands of individual components all working in harmony to process data. Each of these components has its own job to perform; each has its own performance characteristics.

The brain of the system is the Central Processing Unit (CPU), which actually processes all the calculations and instructions that run on the computer. The job of the rest of the system is to keep the CPU busy with instructions to process. A well-tuned system runs at maximum performance if the CPU or CPUs are busy 100 percent of the time.

So how does the system keep the CPUs as busy as possible? In general, the system is made up of different layers, or tiers, of progressively slower components. Because the faster components are typically the most expensive, you must perform a balancing act between speed and cost efficiency.

The following sections look at the performance tiers.

CPU and Cache

The CPU and the CPU cache are the fastest components of the system. The cache is very high-speed memory used to store recently used data and instructions so that it can provide quick access if this data is used again in a short time. Most CPU hardware designs actually have a cache built into the CPU chip. This internal cache is known as a Level 1 (or L1) cache. Typically, an L1 cache is very small—on the order of 8 to 16 kilobytes in size.

When a certain piece of data is wanted, the hardware first looks in the L1 cache. If the data is there, the hardware processes that data immediately. If the data is not available in the L1 cache, the hardware looks in the L2 cache, which is external to the CPU chip. The L2 cache is located very close to the CPU chip because speed is important. The L2 cache is connected to the CPU chips on the same side of the memory bus as the CPU. To get to main memory, you need to use the memory bus, which affects the speed of the memory access.

Although the L2 cache is twice as slow as the L1 cache, it is usually much larger. Its larger size provides a better chance of getting a “cache hit.” Typical L2 caches range from 128K to 4M in size.

Slower yet is the speed of the system memory. It is probably five times slower than the L2 cache. The size of system memory can range from 4M for a small desktop PC up to 2 to 4 gigabytes for large server machines. Some supercomputers have even more system memory than that.

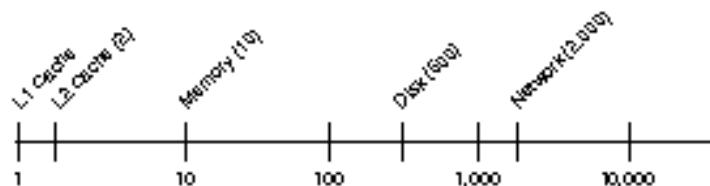
If data or instructions have not been found in any of the system caches or the system memory, an I/O is done through the I/O bus. Any I/Os generated outside of system memory are at least 50 times slower than access from memory. Depending on the storage device and the data being accessed, the access time may even be slower than that.

If data is not available on the system's local disks and must be acquired through a local area network (LAN), accesses are even slower. In the world of the CPU, cache, and memory, it takes an eternity to get data from over a network.

Suppose that it takes 1 second to get a piece of data from the L1 cache in the CPU chip. It takes approximately 2 seconds to get that data from the L2 cache. If the data is not available in the L2 cache and must be retrieved from main system memory, it can take up to 10 seconds to retrieve that data. If the data is not in system memory and you have to go to disk, it can take from 8 to 10 *minutes* to retrieve the data. If you have to get that data from another machine on the network, it can take up to a half *hour* or more (see Figure 13.1).

Figure 13.1

Powers of 10.



As you can see from the time line, there is an enormous difference between retrieving data from the L1 cache and retrieving data from the disk. This is why we spend so much time trying to take advantage of the SGA in memory. This is also why hardware vendors spend so much time designing CPU caches and fast memory busses.

Now look at some of these components and at the different types of computers and architectures.

CPU Design

The CPU is the brains of the computer. This is where most of the instruction processing happens. Although some intelligent devices such as disk controllers can process some instructions, the instructions these devices can handle are limited to the control of data moving to and from the devices.

The CPU works off of the system clock and executes instructions based on clock signals. The clock rate and type of CPU determine how fast these instructions are executed.

The CPU usually falls into one of two groups of processors: Complex Instruction Set Computer (CISC) or Reduced Instruction Set Computer (RISC), as described following.

CISC Processors

CISC processors (such as the type of processors Intel builds) are by far the most popular processors sold today. CISC processors are more traditional and offer a large instruction set to the program developer. Some of these instructions can be quite complicated; most instructions require several clock cycles to complete.

CISC processors are very complex and difficult to build. Because these CPU chips contain millions of internal components, the components are extremely close together. The physical closeness causes problems because there is no room for error. Each year, technology allows more complex and faster chips to be built but eventually, physics will limit what can be done.

CISC processors carry out a wide range of tasks and can sometimes perform two or more instructions at a time in parallel. CISC processors perform most tasks such as RDBMS processing very well.

RISC Processors

RISC processors are based on the principle that if you can reduce the number of instructions the CPU processes, the CPU can be simpler to build and can run faster. By putting fewer internal components inside the chip, the speed of the chip can be turned up.

Remember that it is the system compiler that determines what instructions are executed on the CPU chips. When the number of instructions was reduced, compilers were written to take advantage of this fact and compensate for the missing instructions.

By reducing the instruction set, RISC manufacturers have been able to push the clock speed up many times that of CISC chips. Although the faster clock speed is beneficial in some cases, it offers little improvement in other cases. One effect of a faster CPU is that the surrounding components such as L2 cache and memory must also run faster—at an increase in cost.

Another goal of some RISC manufacturers is to design the chip so that the majority of instructions complete within one clock cycle. Some RISC chips today can already do this. But because some operations that are a single instruction for a CISC chip may be many instructions for a RISC chip, a speed-to-speed comparison cannot be made.

CISC versus RISC

I think that both types of processors have their advantages and disadvantages. It is up to you to decide whether a RISC processor or a CISC processor will do the best job in your implementation.

When comparing the two types of processors, be sure that you look at performance data and not just at clock speed. Although the RISC chips have a much faster clock speed, they actually do less work per instruction. The performance of the system cannot be determined by clock speed alone.

Multiprocessor Systems

Multiprocessor systems can provide significant performance with very good value. With a multiprocessor system, you can start out with one or two processors and add additional processors as your business needs grow. Multiprocessors fall into several categories; two of the main types of multiprocessor systems are the Symmetric Multiprocessor (SMP) systems and the Massively Parallel Processing (MPP) systems.

SMP Systems

Symmetric Multiprocessor (SMP) systems usually consist of a standard computer architecture with two or more CPUs that share the system memory, I/O bus, and disks. The CPUs are called *symmetric* because each processor is identical to any other processor in terms of function. Because the processors share system memory, each processor looks at the same data and the same operating system. In fact, the SMP architecture is sometimes called a *tightly coupled architecture* because the CPUs can even share the operating system.

In the typical SMP system, only one copy of the operating system runs. Each processor works independently by taking the next available job that is ready to run. Because the Oracle architecture is based on many processes each working independently, you can see great improvement by adding additional processors.

The SMP system has these advantages: it is cost effective, high performing, and easily upgradable. Upgrades consist simply of adding an additional CPU to the system to instantly and significantly increase performance. A typical SMP system supports from four to eight CPUs. Because the SMP system shares the system bus and memory, only a certain amount of activity can take place before the bandwidth of the bus is saturated. To add more processors, you must go to an MPP architecture.

MPP Systems

Massively Parallel Processor (MPP) architecture systems are based on many independent units. Each processor in an MPP system typically has its own resources such as its own local memory and I/O system. Each processor in an MPP system runs an independent copy of the operating system and its own independent copy of Oracle. An MPP system is sometimes referred to as a *loosely coupled architecture*.

You can think of an MPP system as a large cluster of independent units that communicate through a high-speed interconnect. As with SMP systems, as you add processors, you will eventually hit the bandwidth limitations of the interconnect. However, the number of processors with which you hit this limit is typically much larger than with SMP systems.

If you can divide the application among the nodes in the cluster, MPP systems can achieve quite high scalability. Although MPP systems can achieve much higher performance than SMP systems, they are less economical: MPP systems typically are much higher in cost than SMP systems.

CPU Cache

Regardless of whether you use a single-processor system, an SMP system, or an MPP system, the basic architecture of the CPUs is similar. In fact, you can find the same Intel processors in both SMP and MPP systems.

As you learned earlier in this chapter, the system cache is very important to the system. The job of the cache is to allow quick access to recently used instructions or data. A cache is always used to store and retrieve data faster than the next level of storage (the L1 cache is faster than the L2 cache; the L2 cache is faster than main memory; and so on).

By caching frequently used instructions and data, you increase the likelihood of a cache hit, which can save precious clock cycles that otherwise would have been spent retrieving data from memory or disk.

System Memory Architecture

The system memory is basically a set of memory chips, either protected or not protected, that stores data and instructions used by the system. System memory can be protected by parity or a more sophisticated advanced ECC correction method. The system memory can range in size from 4 megabytes on a small PC to 4 gigabytes on a large SMP server.

Typically, the more memory available to Oracle, the better your performance. Allocating a large SGA allows Oracle to cache more data, thus speeding access to that data.

System memory is accessed by the CPUs through a high-speed bus that allows large amounts of data and instructions to be moved from the CPU to L2 cache very quickly. Typically, data and instructions are read from memory in large chunks and put into the cache, anticipating that the next instruction or data in that chunk is likely to be accessed next. This process is known as *prefetching*.

Depending on the specific implementation of an SMP system, the memory bus may be shared by all system processors; alternatively, each processor may have a private bus to memory.

The amount of memory available to processes may be limited by the physical memory in the system; more is available if you are using a *virtual memory system*.

Virtual Memory System

In a virtual memory system, the OS and hardware allow programs and users to use more memory than is actually available in the system hardware. This memory is known as *virtual memory*, a large amount of memory that can be mapped to physical memory. Virtual memory must be in physical memory to be used; it is copied out to disk if it is not being used and the space in physical memory is needed. The process of mapping virtual memory onto physical memory by copying the memory to and from disk is called *paging*, or *swapping* (depending on the OS architecture).

Paging and swapping serve the same purpose but operate slightly differently. In a swapping system, an entire *process* is swapped-out (moved from memory to disk) or swapped-in (moved from disk to memory). In a paging system, the movement of data to and from the secondary storage is done on a *memory page* basis; when more memory is needed, one or more pages are paged-out (moved from memory to disk) to make room. If data is requested from virtual memory and is not in physical memory, that data is paged-in (moved from disk to memory) as needed. The rest of this section uses the term *paging* to describe both paging and swapping.

Suppose that you have a computer system with 16M of physical memory. If you have a program that needs to access 20M of data, it is obvious that the process won't fit in physical memory. In a virtual memory system, the data is read in until there is very little memory left (the OS reserves some for itself); then the OS copies some of the data pages to disk with the paging mechanism. Usually, this is done using a *least recently used* algorithm in which the oldest data is moved out. When some memory has been freed, the program can read more data into memory. As far as the program is concerned, all the data is still in memory. In fact, it is—in virtual memory. As the program begins to reread some of the data and manipulate it, different pieces may be paged-in (from disk to physical memory) and paged-out (from physical memory to disk).

As you can imagine, the process of paging-in or paging-out can be quite time consuming and uses a lot of system resources. This is why you have been warned several times in this book not to use so much memory that you cause paging or swapping. Access to disk is approximately 50 times slower than an access to memory.

Bus Design

There are so many bus designs in use today that I cannot go into much detail. A *bus* is a connection path used by the system to move data from one place to another. Although this description sounds simple at first, when you look at it from a performance perspective, you encounter things like the capacity, or bandwidth, of the bus.

The term *bandwidth* is used to describe the amount of data that can be transmitted across the bus in a certain time. Bandwidth was originally used to describe the electronic characteristics of a circuit. Over the years, this term has been adopted by computer designers and is often used to describe the capacity of the bus in megabytes per second (or some other metric).

Several bus designs have been introduced in the last few years, all with the same goal: increased capacity. As processors, network hardware, disk controllers, and disks become increasingly fast, it is necessary for the bus to support the load generated by these faster devices.

You really don't have to worry about the bus under most circumstances. As computers increase in performance, computer designers are taking that performance into account; the system bus should not be a bottleneck in your system.

Summary

This chapter looked at how the system processors work and how performance is affected by various aspects of the computer hardware architecture. You read about the gaps in speed between various components of the system: as you move away from the CPU, components get progressively slower. By taking advantage of system memory and cache through careful tuning of the shared pool, database block buffers, and so on, you can maximize the use of these faster components.

The chapter finished with a brief overview of the system memory architecture and system bus architecture. This overview should give you more insight into the internal workings of your computer and can help you understand why optimizing the use of the SGA is so important.

Chapter **4**

Advanced Disk I/O Concepts

This chapter describes the internal operations of a disk drive. The description is followed by the calculations you need to determine the number of I/Os per second your system can support. These numbers are very important in putting together the system configuration.

In many large systems, the number of I/Os per second that the system can support may be determined by a bottleneck in a single drive. If the RDBMS is waiting for a specific piece of information before it can proceed with the transaction, any time spent waiting (called *latency*) can affect the performance of the entire transaction.

A disk drive has certain limitations you cannot exceed. Once you reach this limit, you cannot achieve a higher I/O rate and performance suffers. As you see in this chapter, you can overcome this problem by adding more or faster disk drives. By understanding these limitations, you will have the knowledge necessary to optimize the use of these disks to achieve the best performance possible.

Disk Operation

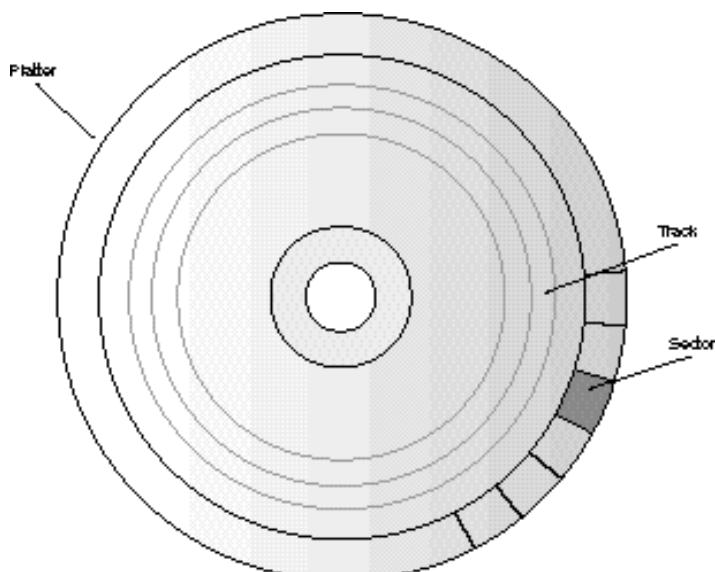
A disk drive is an electronic storage device that holds information. The disk holds that information on the disk magnetically. To read the disk, the input device (*head*) moves over the place where the information is held and determines how the information has been stored. Depending on the magnetism of the spot on the disk, the value of the data is either 0 or 1.

Consider the individual components first and then look at the entire disk drive as it works together to store and retrieve information. The disk drive is made up of thin disks of magnetic storage material called *platters*. These platters look very much like the compact discs (CDs) with which you are familiar.

The platter is divided into *tracks*. A track is the path on which the disk stores information (see Figure 14.1). Tracks are further separated into *sectors*; it is in these sectors that the data is actually stored. There may be 512 or 1,024 bytes per sector, depending on the drive manufacturer.

Figure 14.1

A disk platter.



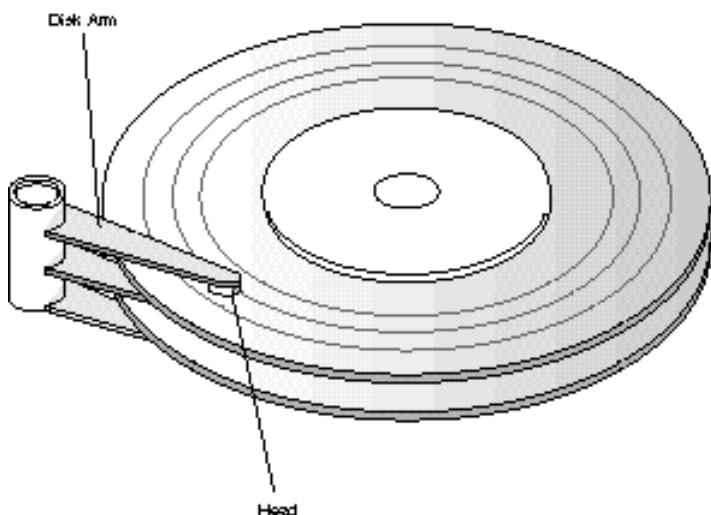
You may have one or more platters in your disk drive. Evolving disk drive technology has allowed manufacturers to pack the tracks tighter, allowing more information to be stored on a platter. The total disk space you have is the product of the number of bytes per platter times the number of platters.

The disk platters rotate at anywhere from 4,000 to 8,000 RPM (revolutions per minute). As the platter rotates, the heads move in and out on a linear path (see Figure 14.2). The heads are attached to a *disk arm* that is used to move the heads in and out.

Usually, the heads and arms are connected so that when the arm moves, all the heads move

Figure 14.2

The disk arm and heads.

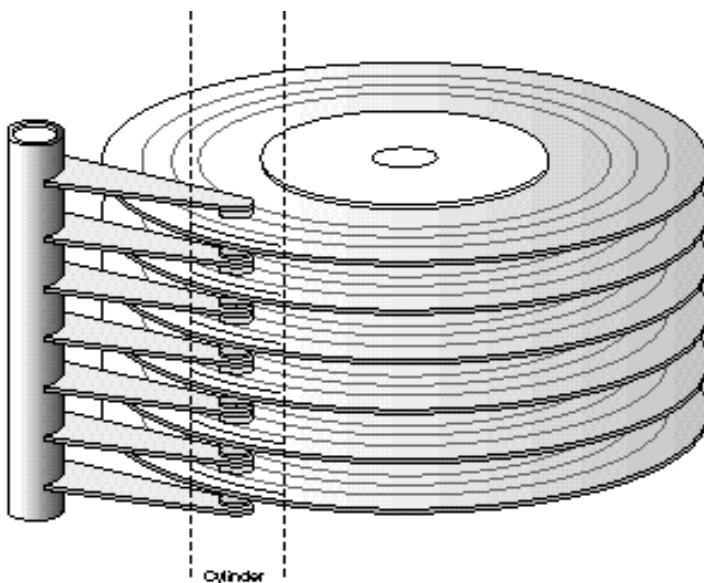


simultaneously. The heads are moved by a small, very precise motor. As information is requested, the disk arm moves the head over the desired track; the disk then simply waits for the desired data to move underneath the head. In this manner, all data is retrieved from the disk.

A *cylinder* is a stack of vertically aligned tracks. Because all the heads are connected, they are all over the same track on different platters. The set of tracks on top of which the heads are located is really what is called the cylinder (see Figure 14.3).

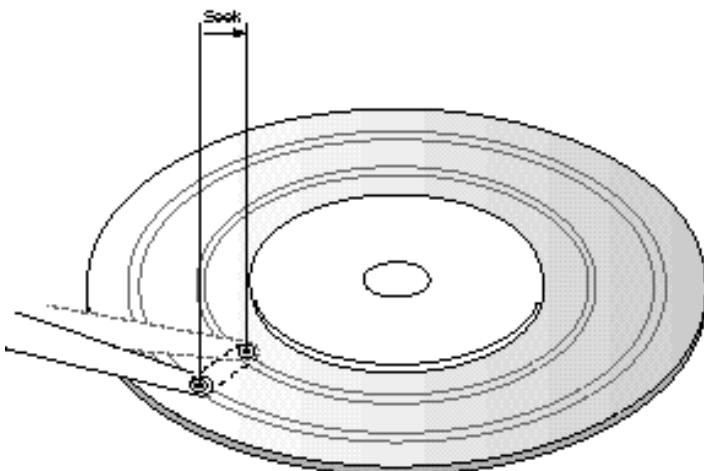
A motor is dedicated to spinning the platters. This motor must be very precise. The final component in the disk drive is the electronics used to control the motors and move the head to the correct place on the disk.

Usually, the disk drive also has a small buffer used to move data in and out of the disk more efficiently. This buffer can be used to sort multiple data requests to try to reduce seek time.

Figure 14.3*Disk cylinders.*

Seek Time

A *seek* refers to moving the head to the proper track. The *seek time* is the time it takes for the heads to move over to that proper track (see Figure 14.4). If you look at the specifications for your disk drive, you will most likely find specifications for single-track, average, and full-stroke seek times. These statistics give you an idea of approximately how long it takes to get to your data. *Single-track seek time* means the time it takes to move to the next adjacent track. *Average seek time* means the average time you can expect the seek to take. *Full-stroke seek time* means the time it takes to seek from the first track to the last track.

Figure 14.4*A disk seek.*

For a typical disk drive available today, the average seek time is from 8 to 15 milliseconds. The single-track seek time is from 3 to 5 milliseconds and the full-stroke seek time is around 15 to 25 milliseconds.

The seek time is very important; it is the seek time and the rotational latency time that really add up to all the time you spend getting to the data. Once you have actually found the data, it takes very little time to transfer that data to memory.

Rotational Latency

Once you move the heads to position them on the proper track, you may have to wait for the platter to rotate until the actual sectors in which you are interested are under the head. The time you spend waiting for the platter to rotate is called the *rotational latency*.

Suppose that your disk drive has a rotational speed of 4,500 RPM. Its number of revolutions per second can be calculated as follows:

$$\text{Revolutions per second} = 4500 \text{ revolutions per minute} / 60 \text{ seconds per minute} = 75 \text{ revolutions per second}$$

Once you know the number of revolutions per second, you can do a simple inversion and determine the number of seconds per revolution:

$$\begin{aligned}\text{Seconds per revolution} &= 1/75 \text{ revolutions per second} = 0.0133 \text{ seconds per} \\ &\text{revolution} = 13.33 \text{ milliseconds per revolution}\end{aligned}$$

Now you know how long it takes for a complete revolution to take place. (Some high-speed SCSI disks have a rotational speed of 7,200 RPM, which results in a rate of 8.33 msec per revolution!) You can guess that the average rotational latency is approximately half a revolution:

$$\text{Average Rotational Latency} = 13.33 \text{ msec per revolution} / 2 = 6.66 \text{ msec}$$

For the example of the disk drive with a rotational speed of 4,500 RPM, the average rotational latency is 6.66 milliseconds. This data is just another piece in the equation used to determine how many I/Os per second your disk drive can handle.

Data Transfer Rate

The data transfer rate should be given in the specifications for your disk drive. The data transfer rate is given in megabytes per second. Typically, the specifications provide values for both maximum and sustained data rates. With many disks, you can hit a maximum data transfer rate of over 15M per second but, in reality, the average sustained rate cannot achieve that. You can approximate the average sustained rate to be 75 percent of the maximum rate. In the disk-drive example begun in the preceding section, assume that the maximum rate is 10M/sec; the

sustained rate is 7.5 M/sec. For the example drive, the data transfer rate in seconds per mega-byte is determined as follows:

Seconds per MB = 1 / 7.5 MB/sec = 0.133 seconds per MB or 0.133 milliseconds per KB

Because a typical Oracle data block is 2K in size, the time it takes for a database block to be read from the disk drive is determined as follows:

Seconds per db block = 0.133 milliseconds per KB * 2 KB per block =
0.27 milliseconds per block

As you can see, relative to the time it takes to do a seek, the time it takes to actually transfer the data is small.

Queue Time

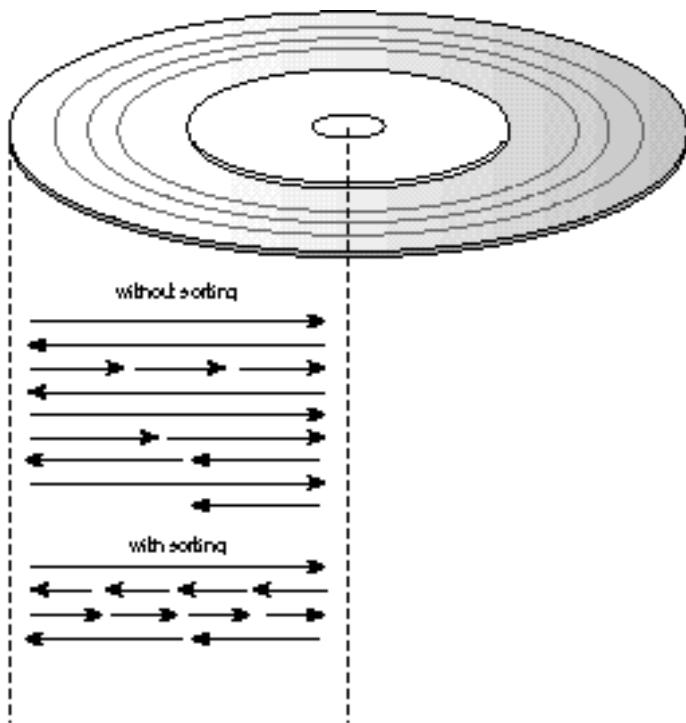
Up till now, all the values given in the example assume that nothing else is happening in the system and that you can get immediate access to the disk. Although many times this is true, there are many other times when other jobs on the system are using the disk and your I/O request must wait. This waiting in line is called *queuing*.

The time you spend in the queue is determined mainly by the OS and the load on the system. Depending on the hardware and OS, the I/O requests may be sorted within the device driver or on the controller (to reduce seeking). Instead of the heads moving back and forth across the disk in a completely random pattern, the sorted requests allow the heads to move more smoothly back and forth across the disk, accessing data in order, based on where on the disk the data is located. Although sorting the data requests quite often reduces seek times, your I/O requests may spend more time in the queue.

This type of sorting is typically referred to as *elevator sorting*. Think of how an elevator works: If you get in an elevator at the 2nd floor and want to go to the 10th floor, the elevator will let you off at the 10th floor before going to the 20th floor, even if someone pressed the button for the 20th floor before you requested the 10th floor. It is much more efficient for the elevator to let somebody off on the floor it is passing rather than in the order the floors are requested. This is similar to what happens to disk I/O requests (see Figure 14.5).

When disk requests are sorted, the disk heads move in a more organized and smoother manner, thus increasing the overall throughput of the disk request. Although increasing this throughput can sometimes reduce the access time for some requests, overall, performance time is increased.

The time you spend in the disk queues varies. The more activity on the disk, the longer the queues. When you reach the point at which where you cannot sustain the I/O rates requested, the queue time begins to account for more and more of the total I/O response time.

Figure 14.5*Elevator sorting.*

Disk Performance

The time it takes for a disk transfer to occur is the sum of the seek time, the rotational latency, and the transfer time. The following sections look at a few examples and characterize some disk drives. First, however, here's a summary of the values discussed so far in this chapter:

<i>Measurement</i>	<i>Average Time in Milliseconds</i>
Average seek time	8 to 15 msec
Single-track seek time	3 to 5 msec
Full-stroke seek time	15 to 25 msec
Seconds per revolution	8 to 15 msec
Average rotational latency	4 to 8 msec
Seconds per database block transfer	0.1 to 0.4 msec

NOTE: The information in the preceding chart is derived from a typical “off-the-shelf” disk drive.

Consider this example: You want to find out how many I/Os per second a disk can sustain. Of course, that answer depends on whether the I/O is random or sequential. The following sections describe random I/Os (as are used in data files) and sequential I/Os (such as those used in the redo log files).

Random I/Os

The example in this section determines how many random I/Os per second you can sustain on a data drive. For this example, use the following values:

Average seek time	14 msec
Average rotational latency	5 msec
Data transfer time (1 database block: 2,048 bytes in size)	0.27 msec
Queue time	0 msec

This formula calculates how long it takes to retrieve a block of data on average:

$$\text{Avg. Seek Time} + \text{Rotational Latency} + \text{Data Transfer Time} + \text{Queue Time}$$

For the current example, the data retrieval time is calculated as follows:

$$14 \text{ msec} + 5 \text{ msec} + 0.27 \text{ msec} + 0 \text{ msec} = 19.27 \text{ msec}$$

What does this tell you about the number of I/Os per second the disk can sustain? Because you are not taking into account disk queuing and elevator sorting, this calculation is somewhat crude but it does give a ballpark figure. If you assume that the disk can return a 2,048K database block in 19.27 msec, the number of I/Os the disk can sustain is determined as follows:

$$\begin{aligned} \text{I/O requests per second} &= 1 / \text{second per I/O request} = 1 / 19.27 \text{ msec per I/O} = \\ &1 / 0.01927 \text{ sec per I/O} = 52 \text{ I/Os per sec} \end{aligned}$$

By this crude calculation, the sample disk drive can roughly support 50 random I/Os per second per drive. This is consistent with information you have seen before (and now you know roughly how this number was derived).

When you start approaching this limit, other effects (such as disk queues) have greater influence on the access time.

Sequential I/Os

Now look at the case of purely sequential I/O. Sequential I/O has a much better performance rate than random I/O because there is very little head movement involved in sequential I/O.

This example looks specifically at the I/O rates of the log writer. For this example, assume the following specifications:

<i>Measurement</i>	<i>Average Time in Milliseconds</i>
Average seek time	14 msec
Single-track seek time	4 msec
Average rotational latency	5 msec
Data transfer time (1 database block: 2,048 bytes in size)	0.27 msec
Queue time	0 msec

For this example, look at how long it takes to retrieve a block of data on average. For sequential I/O, the access time is determined by a much smaller number of seeks. The number of times you have to seek is determined by when you have to go to the next track on the disk. Here are some additional specifications for the example drive:

Sectors per track 63
Number of platters 3

Because disk technology is very efficient, in a sequential I/O, the heads write or read the data to the same cylinder before moving on. For a disk with 63 sectors per track and 3 platters with 2 sides each, the total amount of data that can be written or read before a single track seek has to occur is determined as follows:

`63 sectors/track * 3 platters * 2 tracks/platter * 512 bytes/sector = 193,536 bytes
= 189 Kbytes`

Because the log writer writes the data in chunks of `LOG_BUFFER_SIZE` size (which is set to four times the size of the database block size), you can say that approximately 13 blocks are written before a seek must occur. This formula approximates the average time spent seeking:

`Average Seek Time = Single-Track Seek Time / 13 = 4 msec / 13 = 0.3 msecs`

As you can see, the average seek time does not come into play at all with sequential I/Os. But what about the rotational latency? There *is* a problem there. Because the log writer fills the log buffer and then writes it out in chunks, it is unlikely that the disk heads will always be over the proper place on the disk. Therefore, rotational latency is still relevant for sequential I/O.

Now look at the sequential performance in terms of the number of I/Os per second the disk can sustain. Because the I/Os you are looking at now are 8K in size (because this is the size of the log buffer), the data transfer rate is calculated as follows:

`Seconds per log buffer = 0.133 milliseconds per KB * 8 KB per buffer =
1.06 milliseconds per block`

Here's the formula to determine the data transfer rate:

Average Seek Time + Rotational Latency + Data Transfer Time + Queue Time

For this example, the data transfer rate works out like this:

$0.3 \text{ msec} + 5 \text{ msec} + 1.06 \text{ msec} + 0 \text{ msec} = 6.36 \text{ msec}$

Therefore, the sequential I/O rate for this particular disk drive is calculated as follows:

$\text{I/O requests per second} = 1 / \text{seconds per I/O request} = 1 / 6.36 \text{ msec per I/O} =$
 $1 / 0.00636 \text{ sec per I/O} = 157 \text{ I/Os per sec}$

Now assume that the log writer can use Asynchronous I/Os and can put enough requests in order to reduce the rotational latency. In that case, the number of I/Os increases significantly. The best possible case is one in which the log writer always writes enough data so that rotational latency is eliminated. You can do this by increasing the size of the log buffer or by enabling Asynchronous I/O on the LGWR. In this case, the maximum theoretical I/O rate is determined by using the following calculation:

Average Seek Time + Rotational Latency + Data Transfer Time + Queue Time

For this example, the calculation is as follows:

$0.3 \text{ msec} + 0 \text{ msec} + 1.06 \text{ msec} + 0 \text{ msec} = 1.36 \text{ msec}$

Then use this formula to calculate the number of sequential I/Os per second:

$\text{I/O requests per second} = 1 / \text{seconds per I/O request} = 1 / 1.36 \text{ msec per I/O} =$
 $1 / 0.00136 \text{ sec per I/O} = 735 \text{ I/Os per sec}$

This calculation gives the theoretical maximum number of sequential I/Os per second per disk as 735 I/Os per second per disk. Remember that these calculations are crude and don't take into account other factors that come into play as you near the limits of the capacity of the drives (factors such as queues and disk buffers).

What is important to remember is the difference in performance you can achieve with sequential I/O versus random I/O. This performance difference is why it is so important to separate the log disks from the data disks. Because the redo log is so important to the operation of the RDBMS, you want to make sure that you can log as fast as necessary. Any delays in writing to the log volumes are reflected throughout the entire system.

Summary

This chapter explained how the physical disks operate and provided some calculations to show how the magic numbers on disk I/O rates are calculated. I want to reemphasize that these numbers are very rough and should be used as general guidelines.

The actual I/O rates you achieve depend not only on the physical disks themselves but on the disk controllers, the bus, and the operating system. As newer disks are being developed, new features are being added to improve performance.

It is important to note that the number of random I/Os you can achieve system wide depends on the number of disks you have in the system. Remember this when you are configuring your next system: Although disk drives are getting larger and faster, the increased speed does not compensate for the performance you can get with more disks.

It is usually much better to get more smaller disks than fewer larger disks. For example, a 2 gigabyte SCSI disk with a rotational speed of 4,500 RPM and an average seek time of 10 msec can achieve approximately 60 I/Os per second per disk. A 4 gigabyte SCSI disk with a rotational speed of 7,000 RPM and an average seek time of 8 msec can achieve approximately 79 I/Os per second per disk. However, two 2 gigabyte drives can support 120 random I/Os per second (compared with only 79 I/Os per second for the 4 gigabyte drive).

The next chapter takes these concepts a bit farther when you look at disk arrays.

Chapter

5

Disk Arrays

A *disk array* is collection of disk drives that are configured and act as one larger disk drive. Both hardware and software disk arrays are available today. Hardware disk arrays consist of a disk array controller and a set of disk drives (typically SCSI). Software disk arrays are made up of a software layer that lies between the file system and the device driver layers.

Disk arrays usually support disk striping, disk mirroring, disk parity, and so on. Hardware disk arrays also support other features such as hot swappable disks. Depending on the manufacturer and model, the disk array controller can support several different types of RAID fault tolerance.

Disk arrays offer many benefits to the system administrator and the end users:

- ◆ **Ease of management.** A disk array can offer tens of gigabytes of disk space that appear to the administrator as one large disk. This arrangement simplifies some of the management tasks involved with managing large numbers of disks.
- ◆ **I/O balancing.** Because a disk array is made up of many individual disks with striped data, random I/Os are automatically distributed among the disks.
- ◆ **Fault tolerance.** Disk arrays provide a wide range of RAID options with a wide variety of performance and economic choices.

This chapter examines the operation of disk arrays. You will see how a disk array works and what features are available. You will look at the different fault tolerant modes and examine the different RAID levels available in today's disk array controllers.

The end of the chapter looks at how you can take advantage of disk arrays from a performance standpoint. How do you best configure your database to use disk arrays? How are disk arrays different from standard disk subsystems?

As this chapter looks more into the workings of the disk array, you will understand why disk arrays are so popular today. This chapter helps you determine whether your system can benefit from a disk array or whether you can improve your system by reconfiguring an existing disk array.

How Does a Disk Array Work?

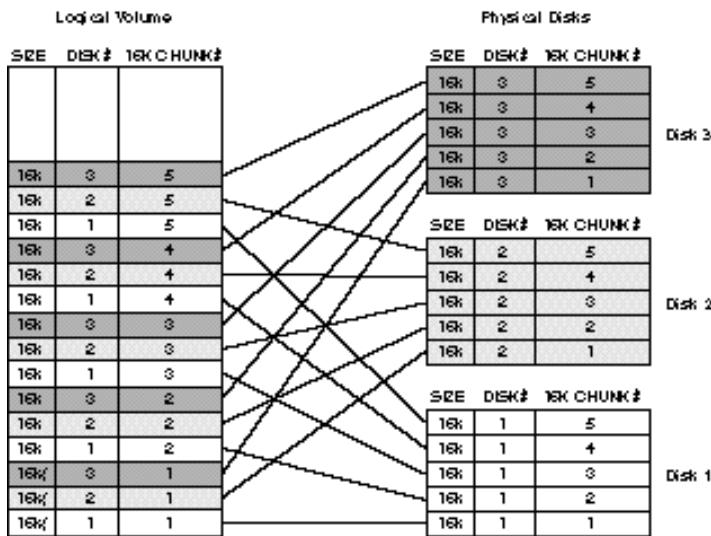
A disk array is a set of disk drives that make up a larger logical disk, sometimes called a *logical volume*. The logical volume is made up of identical-sized pieces of the individual drives, called *stripes*. The data is said to be striped across the logical volume because the logical drive has pieces of all the individual drives striped within it (see Figure 15.1). If you look at the logical volume, the physical drives seem to make stripes.

The stripes are all the same size and in sequence (that is, in the order of the disk and the position on each individual disk). The size of the stripe varies based on the manufacturer and model of the disk array controller. The stripe size also varies in a software array. The size of the stripe is called the *striping factor* and is given in number of blocks. A typical striping factor is 32. Given a 512 bytes-per-block size, a striping factor of 32 makes the disk stripe 16K in size.

As far as the user and the RDBMS is concerned, there is only one large disk drive. It is up to the OS and the hardware to manage the individual disks within the array.

Figure 15.1

Disk stripes on a logical volume.



Software Array

In a software array, all the disk-array processing is done by the system CPU. Each time a disk request is issued, the OS must perform all the processes needed to maintain the array. If fault tolerance is used, significant overhead can be involved.

The extra processing involved in maintaining a software disk array can be a performance problem if you are running your machine near the capacity of the CPUs. If you have a significant amount of idle CPU cycles, the overhead associated with a software array will not be noticeable.

Hardware Array

In a hardware disk array, all the processing associated with maintaining the array is done on the disk controller itself. The disk controller has its own CPU, which is responsible for performing the necessary processing. Even when you employ fault tolerance, the CPUs on the system are not affected; all the fault tolerance processing is done on the controller itself.

The controller also includes a small amount of memory to use in this processing. Most hardware disk array controllers have enough CPU power to keep up with any I/O rate requested of it. The only bottlenecks occur when the I/O rates of the physical disk drives that make up the array are exceeded and jobs start queuing up.

Sophisticated disk array controllers also offer advanced features such as hot swappable disk drives, automatic rebuild options, and advanced fault detection and correction. Fault correction is available only if you run in a fault-tolerant mode.

Hot-Swappable Disk Drives

Many vendors support hot-swappable disk drives with their disk array products. This type of drive allows failed disk drives to be removed and replaced without having to shut down or power off the system. Most disk array controllers can automatically rebuild a disk drive that has replaced a failed drive. This feature works in conjunction with advanced fault detection features that continually test and report failed or “soon to fail” disk drives.

A disk array such as the Compaq SMART disk array can detect faults and signal a remote monitoring station. The faulty disk drive is indicated by a red light on the drive itself. If this array is running in a fault-tolerant mode, an operator can simply remove the faulty disk drive and replace it with a spare. The spare drive is automatically rebuilt using information stored on the remaining disk drives without operator intervention. The rebuild is done while the drives are still online and fulfilling user requests. Only a small degradation in performance is experienced while the disk drive is being rebuilt. You can even configure an optional online spare drive that instantly and automatically replaces a faulty drive. The online spare drive replaces a faulty drive the moment the fault is detected.

Array Controller Caches

Many disk arrays on the market today offer a large controller cache. This cache can help in several ways:

- ◆ **Cache hits.** If the data being accessed is frequently written and read, you may see some benefits from a hit in the controller cache.
- ◆ **Fault tolerance.** The write-caching of the data hides the effects of fault tolerance. Fault tolerance incurs a certain amount of overhead, as described in “RAID Technology,” later in this chapter. The write-cache can compensate for this overhead.
- ◆ **Disk queuing and sorting.** By queuing the I/O requests on the controller rather than in the OS, the I/O requests can be sorted based on the disk on which the data is located.

Write Caches

Write caches are designed to cache the writes written to the controller and to post the I/O request to the disk driver. *Posting the request* means that the device driver believes that the I/O is completed. Posting the write allows the overhead associated with queuing the request and writing it to the individual disk to be masked. All the OS and the RDBMS see is an incredibly fast write.

CAUTION: Unless the write cache is protected, it is not usually recommended that write-caching be enabled on the redo log volume. If the RDBMS believes a write to have occurred to the redo log and a controller failure occurs before that data has been written to the disk drives, you may not be able to recover the instance.

Some disk array controllers like the Compaq SMART array have incorporated both mirroring and a battery backup into the write cache to ensure that the data is protected even in the event of a power outage. Use your own judgment about running a write cache on the redo log volumes.

Read/Write Caches

Controller read/write caches are similar to write caches (described in the preceding section) with the exception that during a read, the controller checks the cache first. If the data is in the cache, the data is returned immediately; if not, the request goes to disk. Although reading from cache may help in some situations, it is usually not a great advantage in an RDBMS environment because you already have an enormous read cache: the database block buffers. The chance is very small that an I/O written to the disk controller is in the read cache on that controller but is not in the SGA.

Because you already have a highly optimized cache (SGA) in an RDBMS, it is unlikely that there will be a request to read data that has just been written. The chances are small that this data is in the disk cache but not in the SGA.

However, you will reap the benefits of the write cache if you use fault-tolerant functions. The controller can mask the effects of extra I/Os that may be involved in your fault tolerant method.

RAID Technology

The configuration of disks in an array is sometimes called a Redundant Array of Inexpensive Disks (RAID) configuration. RAID technology has allowed systems to maintain very large amounts of storage at relatively low cost. SCSI drives have not only improved in performance and quality over the years, they have significantly dropped in price.

The term *RAID* is also used to describe the type of striping you use. Striping methods vary in both performance and space overhead. The type of configuration used in your system depends on your individual needs. Different configurations of striping and fault-tolerant striping are identified by RAID levels.

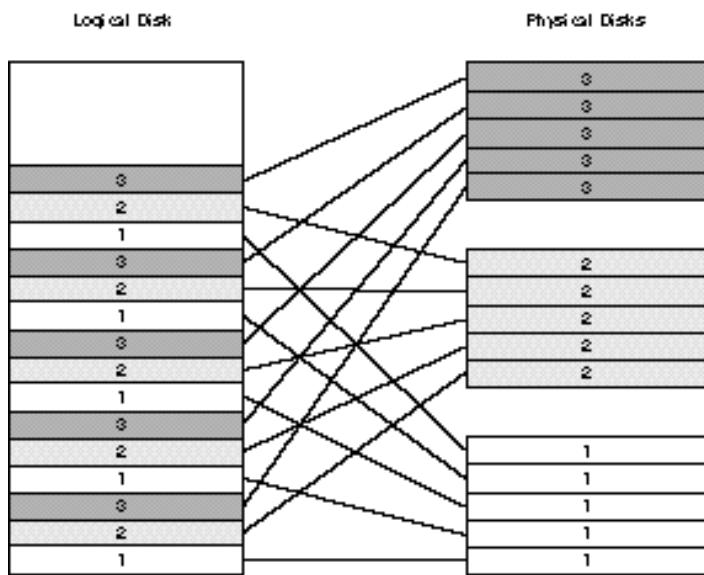
Although you may see various RAID levels advertised by hardware manufacturers, there are really standards for only RAID levels 0 through 5 at this time. And because these standards are based on general concepts and are implementation specific, different implementations of RAID-5 (for example) may use different algorithms. The six RAID levels are described next.

RAID-0

RAID-0 is the base RAID level and is used to describe disk striping with no fault tolerance. RAID-0 drives simply have data striped over all the drives (see Figure 15.2).

Figure 15.2

RAID level 0 has no fault tolerance.



RAID-0 is the highest performing and most economical of the RAID levels. The amount of data in the logical volume is equal to the sum of the amount of data on each disk drive. RAID level 0 causes no additional I/Os to be generated on behalf of the fault tolerant method.

The downside of RAID-0 is that if a disk were to fail, the entire volume becomes invalid. Because the data in the logical volume is striped across all the disks, a loss of a single disk causes a loss of data throughout the logical volume. If a 14-disk volume were to fail, you must restore the data for all 14 disk drives. There is no way to back up and restore data for a single drive in a disk array.

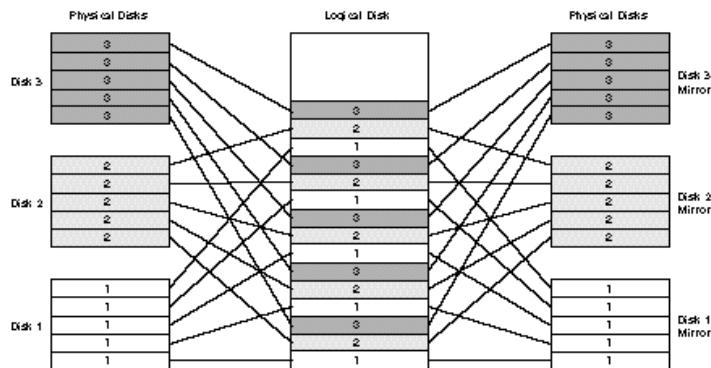
If you are looking for the highest performance and the best value possible—and are not worried about fault tolerance—this is the RAID level you should use.

RAID-1

The RAID-1 level is also known as *disk mirroring*. In RAID-1, all the data stored on a disk drive is duplicated on another disk in the array. Each time a write occurs to the logical disk, the data must be written to *both* physical disks before the logical write is considered completed. With disk mirroring, a single disk is mirrored to another disk; these disks can also be striped with other disks to form a larger logical volume (see Figure 15.3).

Figure 15.3

RAID level 1 is also called *disk mirroring*.



Because the mirroring is on a one-to-one basis, you can actually lose half the disks in your system (depending on which disks they are) and still be operational. With most disk array controllers, you can split the mirror across SCSI busses. This arrangement allows for the failure of an entire SCSI bus (for example, a cabinet failure) without affecting operation.

With disk mirroring, you can use only half the disks in the system (the other half are used for the mirrors). In other words, if you use disk mirroring with two 2.1 gigabyte disks, you can use only 2.1 gigabytes of space. When writing, you get the benefit of only one disk in terms of performance because a logical write invokes two physical writes.

You may see a benefit from reading from a mirrored drive. Some disk array controllers support *split-reads*, in which reads can occur simultaneously on different mirrored drives to different data. The disk with the heads closest to the requested data retrieves the data. Depending on the data access methods, this feature may or may not provide any benefits.

If you can afford the cost of disk mirroring, RAID-1 is the best choice when fault tolerance is required. With disk mirroring, you achieve the highest level of protection as well as the fastest disk access possible for a fault-tolerant volume.

RAID-2

RAID level 2 is data striping done at the bit level. It is a theoretical concept only; no manufacturer has yet built a RAID-2 disk controller.

RAID-3

RAID level 3 is sector striping. As is true for RAID-2, RAID-3 is considered a theoretical concept.

RAID-4

RAID level 4 is known as *drive parity*, or *data guarding*. In RAID-4, one of four drives is used for data parity. If any one of the four disks fails, the other three continue running. Because the algorithms used in RAID-4 are considered obsolete, RAID-4 is not used much. RAID-4 has essentially been replaced by RAID-5.

When you use RAID-4, many extra I/Os are generated. To calculate the parity, RAID-4 reads data from all the drives in the set. As you can imagine, the overhead involved with RAID-4 is very high.

RAID-5

RAID-5 is also known as *distributed data guarding*. In distributed data guarding, enough information is stored about each drive so that any one drive in the set can fail; the data is restored by using all the other drives. The space you get from RAID-5 is equal to the following:

$$\text{Drive Space} = (N - 1) * \text{Drive Size}$$

In this equation, N is the number of drives in the logical volume. In other words, if you are running 12 disk drives with RAID-5, you get the space of 11 disk drives.

As with RAID-4, there is a penalty associated with RAID-5. For each write to disk, two reads take place, a calculation of the parity is done, and then two writes are done. Although a read generates only one I/O, a write generates four I/Os.

Summary of RAID Levels

You can choose from several different RAID levels based on your particular performance and budget needs. Table 15.1 summarizes the performance characteristics of the most popular RAID levels (RAID-0, RAID-1, and RAID-5).

Table 15.1 The I/Os Generated by RAID Fault Tolerance

RAID Level	I/Os per Write	I/Os per Read
RAID-0 (no fault tolerance)	1	1
RAID-1 (mirroring)	2	1
RAID-5 (distributed data guarding)	4 (2 Reads, 2 Writes)	1

Keep in mind that many disk array controllers allow several logical volumes of different types on the same controller. You are not limited to a single RAID level per system or controller. You may find this freedom useful in your configuration.

Fault-Tolerance Concerns

If you use RAID technology, there are essentially three modes in which you can run your database:

- ◆ **No Data Protection.** Fault tolerance is not used on any of the logical volumes.
- ◆ **Full Data Protection.** Fault tolerance is used on all the volumes.
- ◆ **Partial Data Protection.** Fault tolerance is used on some of the volumes and not on others.

Although disk drives have become much more reliable in the past few years, they are still the component most likely to fail. The reason that disk drives are susceptible to failure is because they are mostly mechanical devices.

Disk drives are made up of sophisticated electronics and motors. Forces such as heat, dust, and friction can cause even the most well-built disk drive to fail. No matter how well you take care of your system, it is inevitable that you will one day see a disk failure.

TIP: Depending on the size and critical nature of your database, a failure of one disk can be devastating if you are not protected. When taking into account your budget for fault-tolerant components, keep in mind the cost of extended downtime that results from a failure.

No Data Protection

In No Data Protection mode, no fault tolerance is used and the data is completely unprotected. Any disk failure causes you to reload all the affected files from the last backup.

If the failure occurred on a data volume, you can restore the data (if you are running in ARCHIVELOG mode, you can use the archive log files to restore the database up to the point of failure). Even so, when a disk drive fails, you must restore the entire *volume*—which may mean tens of gigabytes of data.

If the failure occurred on the volume containing your redo log files, you can recover only up to the last good archived log file. At instance recovery, you can recover only up to the point at which the LGWR started writing to the damaged log file(s).

If the failure occurred on the volume containing the OS and Oracle binaries, you must restore from a backup or reinstall the operating system, reload Oracle, and then restart Oracle.

As you can see, in each of these unprotected situations, the recovery interval can be quite long. No failure of an unprotected system is easy to recover from.

Full Data Protection

Full Data Protection is by far the most secure option. If you have fault tolerance on every disk drive in the system, any single failure does not cause a system failure. Depending on the level of fault tolerance you run, many multidisk failures can be tolerated. The level of fault tolerance you run depends on the performance you require and the budget you have.

In the last few years, the cost of disk drives has dropped so dramatically that it is not unusual to see entire systems run with disk mirroring. The cost of the extra drives needed to support fault tolerance usually outweighs the cost incurred by extended downtime.

In assessing the level of fault tolerance, look at your required I/O rates. If possible, use RAID-1 (it gives the best performance). If RAID-1 is not in your budget, perhaps you can use RAID-1 for at least your OS and redo log volumes. The redo log volumes usually demand the highest protection available.

How To Determine I/O Rates

To properly size your I/O subsystem, it is necessary to determine your I/O throughput rates. Unfortunately, this is not an easy task. The required I/O rate is determined by many factors, including the number of users, the number of transactions, the size of the SGA, and so on. It can usually be determined only by measuring the system under load. If you have profiled your application, you may already have some of this information. Once you determine the number of I/Os per second that your system requires, you can then determine the number of disk drives you need.

Partial Data Protection

Another option for protecting your data is to apply fault tolerance to specific pieces of critical data. This may be the best option if you have a limited budget and require high performance.

Critical areas that should be protected are the OS, the redo logs, and the archive logs. If these areas are protected and a drive volume failed, you would not have to spend extra time restoring or reloading the operating system. If the redo log volumes are not protected and a drive failed, you might not be able to recover your database to the point of failure. If the archive files are not protected and a drive failed, you might not be able to recover in the event of drive failure on a data volume.

By using partial fault tolerance, you may be able to reduce downtime in the event of a failure and still keep within your budget. By protecting areas such as the OS volumes and redo log volumes, you can reduce the amount of time needed to recover.

Fault Tolerance Summary

The most likely component in your system to fail is a disk drive because it is a heavily used mechanical device. To avoid extended downtime, it is wise to employ some type of fault tolerance. The extent to which you protect your data depends on your uptime requirements, performance requirements, and budget.

Depending on your needs, you may not have to completely protect all your data—but *some* fault tolerance is recommended. If it is too expensive to protect all your disks, protect at least the critical areas. Depending on your needs, you may want to use different volumes. Here are a few suggestions about fault tolerance for you to consider:

- ◆ Fully protect your OS and Oracle binaries using RAID-1 or RAID-5. This level of protection is worth the cost of the extra disks because you avoid reloading the OS and Oracle.
- ◆ Fully protect the redo log files. If possible, put each redo log file on a separate mirrored volume. By separating the redo log files, the sequential nature of the I/Os generated by the LGWR is maintained even when archiving.
- ◆ Use RAID-5 on read-intensive volumes. The performance degradation experienced when using RAID-5 occurs only on writes. If the volume is mainly used for reads, very good performance can be achieved with RAID-5.

The amount and type of fault tolerance you employ depends on your specific requirements. If you cannot permit any downtime, you must protect yourself against disk failures.

Configuring RAID for RDBMS Performance

RAID volumes can be very useful in an RDBMS system. One key advantage of RAID is its ability to spread I/Os evenly across a large number of disks. If a disk array has a 16K stripe, roughly every 8 database blocks will be on a different disk.

If you do random I/Os, chances are good that the I/Os will be fairly evenly spread across the disks. By creating a disk array with a sufficient number of disk drives, I/Os should not be a problem.

Here are some configuration issues you should consider:

- ◆ **Isolate sequential I/Os.** Even in a RAID configuration, drives are driven sequentially if the I/Os to the logical volume are sequential. Isolate areas such as the redo log files and log archive destination. By separating the sequential I/Os, you can drive the disks used for the sequential volumes at a much higher I/O rate.
- ◆ **Distribute random I/Os.** A larger number of disks in a random volume can support a larger number of random I/Os. The more disks you have in the volume, the more random I/Os you can support.

- ◆ **Size the volumes properly.** An underconfigured volume causes an I/O bottleneck. Don't exceed I/O limitations. In Chapter 14, "Advanced Disk I/O Concepts," you learned how to determine the optimum number of I/Os per second per disk. Be sure to take into account the extra I/Os generated by the disk array for fault tolerance.
- ◆ **Configure for the disk array.** To take maximum advantage of disk array technology, you must abandon some of the configuration guidelines you have used in the past. Some of these new ideas are the complete opposite of what you have done in the past.

Because all these issues are important, the following sections look at each individually.

Isolate Sequential I/Os

Isolating the sequential I/Os is very important—even with a disk array. As you learned in Chapter 14, "Advanced Disk I/O Concepts," a sequential access to a disk reduces the time spent seeking between tracks. The seek time can be a large percentage of the time it takes to do an I/O.

By reducing the head movement, you can get both a higher throughput on the disks and a faster response time—both of which are important in an RDBMS environment. Here is a partial list of the things that are handled sequentially in an RDBMS system:

- ◆ **Log writes.** The LGWR writes to the redo log files in a purely sequential fashion.
- ◆ **Archive log reads and writes.** The archive log process reads from the redo log files sequentially and writes to the archive log files sequentially.
- ◆ **Import/export.** The import process reads from the import file in a sequential fashion; the export process writes to the export file in a sequential fashion.
- ◆ **Database load.** The SQL*Loader utility reads the load file sequentially, writes to the database sequentially, and writes to the index sequentially.
- ◆ **Report files.** In the case of a detailed analysis report, the report file generated can be many megabytes in size and is written sequentially.
- ◆ **Index creation.** Creating an index is somewhat sequential; however, chained and migrated rows break up some of the sequential nature of the table scan.
- ◆ **Decision support activities.** Decision support activities such as supply and demand reporting and market studies typically involve large table scans that are mostly sequential.
- ◆ **Report processing.** Report processing typically involves large table scans that are mostly sequential.

Full-table scans are mostly sequential in nature. Chained and migrated rows cause the sequential nature of the scan to be broken up; so does fragmentation of the database. As these problem areas are reduced or eliminated, the table scans become more sequential.

By isolating the sequential I/Os to a particular disk volume, the performance of that volume can be quite high. However, anything that causes random I/Os on the sequential volumes eliminates those performance gains.

To take best advantage of space, you can use the OS volume as a sequential volume if you determine that the system is used only for RDBMS processing, if very few OS activities produce I/Os on that volume, and if the system does not page or swap (that is, if the page/swap file is on the same volume).

Distribute Random I/Os

You should distribute random I/Os across as many physical disks as possible. Doing so spreads out the load among many disks and enables the disk array to process as many random I/Os as those disks allow. If a single disk in the array can handle 50 random I/Os per second, it is reasonable to assume that an array of four similar disks can handle 200 random I/Os per second.

When using a disk array, you do not have to split different types of data files among different logical volumes. In fact, you should do the opposite. When using a disk array, put all the data files—and their indexes—on the same logical volume. The exception to this guideline is a decision support or batch processing system.

The reason it has always been recommended that indexes and data files be separated was to allow for simultaneous accesses. But simultaneous access is what disk arrays excel at. In fact, the more simultaneous accesses you make to the disk array (until saturation), the better the overall throughput.

TIP: It is still a good idea to separate indexes and data into separate data files for ease of management, but put them both on the same logical volume for performance.

Load balancing of different data files is also unnecessary when using a disk array. It has always been a difficult task for the database administrator to try to balance each data file so that the I/O rates for each disk drive are balanced. Usually, that task is ineffective because of the changing nature of the database workload.

By placing all the data files and indexes across a large striped volume, you are guaranteed to get a better load balance than you can get by hand. In a striped disk configuration, all disks average the same number of I/Os per second, regardless of the configuration and workload.

Earlier, I mentioned that the exception to the rule that data and indexes should be configured on the same logical volume is with the decision support and batch processing systems. Decision support and batch processing typically involve bulk loads and batch updates. A typical scenario is for data to be extracted from the production OLTP systems on a regular basis and loaded onto the DSS (decision support system).

During the database load (which is usually time critical), these four sequential operations may happen simultaneously:

- ◆ Reading from the load file
- ◆ Writing to the redo log files
- ◆ Writing to the data files
- ◆ Writing to the index files

In a DSS or batch processing system, it is a good idea to separate the data and index files on separate logical volumes for load purposes.

The data generated from a detailed report file can also cause a significant amount of sequential I/O. If the batch report is the only job being run on the system, the report can be written to the same volume as the redo log file or to the batch load volume because neither of these are active during the batch processing job.

By configuring a DSS system as just described, you lose some performance when you run queries because the load is not as well balanced as it would have been if the indexes and data shared the same drives. This minor performance loss is well made up for by the fact that loading is significantly faster.

Size the Volume Properly

Make sure that you have enough disk drives to support the load you want to put on the logical volume. In Chapter 14, “Advanced Disk I/O Concepts,” you learned how many sequential and random I/Os per second your disks can support. Now use that information to determine how many disks you need in a disk array.

Suppose that you want to support 250 random I/Os per second to your data files. These I/Os are split into 125 writes per second and 125 reads per second. Now let’s analyze these requirements for several fault tolerant methods.

Recall the data on RAID levels presented earlier in Table 15.1 (presented again here for your convenience).

Table 15.1 The I/Os Generated by RAID Fault Tolerance

<i>RAID Level</i>	<i>I/Os per Write</i>	<i>I/Os per Read</i>
RAID-0 (no fault tolerance)	1	1
RAID-1 (mirroring)	2	1
RAID-5 (distributed data guarding)	4 (2 Reads, 2 Writes)	1

Table 15.2 shows the number of I/Os per second generated for each RAID level when you run at 125 writes per second and 125 reads per second.

Table 15.2 I/Os by RAID Level at 125 Reads per Second and 125 Writes per Second

<i>RAID Level</i>	<i>Writes/sec</i>	<i>Reads/sec</i>	<i>Total/sec</i>
RAID-0 (no fault tolerance)	125	125	250
RAID-1 (mirroring)	250	125	375
RAID-5 (distributed data guarding)	250	125 + 250	675

By using the value of 50 I/Os per second per drive, you can calculate the number of disk drives you need for each of these fault tolerance methods to achieve the required I/O rate (see Table 15.3).

Table 15.3 Number of Disk Drives Required

<i>RAID Level</i>	<i>I/Os per second</i>	<i>Disk Drives Required</i>
RAID-0 (no fault tolerance)	250	5
RAID-1 (mirroring)	375	7.5 round to 8
RAID-5 (distributed data guarding)	675	13.5 round to 14

As you can see from Table 15.3, even though RAID-5 is much more economical in terms of I/Os per second, you may need many more disk drives if you are in an I/O-bound situation. By configuring your system with RAID-5 so that you can achieve the desired number of I/Os, you also get the benefit of additional disk space because you will be using many more drives. Table 15.4 looks at the same data in terms of disk space (assume that you are using 1-gigabyte SCSI disk drives).

Table 15.4 Disk Space by RAID Level

<i>RAID Level</i>	<i>Disk Drives Derived</i>	<i>Disk Space Provided</i>
RAID-0 (no fault tolerance)	5	5 gigabytes
RAID-1 (mirroring)	8	4 gigabytes
RAID-5 (distributed data guarding)	14	13 gigabytes

As you can see, to provide for the extra overhead involved in RAID-5, you may have to purchase more disk drives than you really need for your configuration. Again, choose the RAID level that best meets your needs.

Configure for the Disk Array

As mentioned several times in the first parts of this chapter, you must abandon some of the configuration guidelines you have used in the past to take maximum advantage of disk array technology.

With traditional disks, the database administrator tries to segment data and indexes over different disks to achieve concurrent access to both data and indexes. Furthermore, tablespaces are made to span multiple disks to balance the load on the disks.

This balancing act can be very difficult and sometimes impossible. The outcome usually results in some disks being overworked while others are underutilized. As access to the database changes over time, the “hot spots” can migrate, making the load balancing worse.

When you use a disk array, abandon the old concepts and guidelines and configure the database differently. You should still isolate the sequential and random I/O as you do with traditional disks, but now you should put all the random data and indexes on the same logical volume.

The properties of a disk array allow concurrent access to the drives. In fact, the more I/O requests you make to a disk array, the better. Up to the point at which the disks are saturated, you benefit from more requests. These I/O requests are internally sorted by disk drives and then queued and sorted by the controller to achieve maximum throughput on the disk drives.

The only way to get more performance from a randomly accessed disk is to add more disks. Therefore, the most effective way to improve system-wide I/O performance is to supply as many disks as possible for your random requests. Do this by putting all the random data on the same logical volume and putting as many disks as possible on that logical volume.

The exception to this rule is the DSS or batch processing system that can benefit from the sequential nature of loads to both the data and index files. By separating the data and indexes, you can greatly reduce the load time.

RAID Comparison

This chapter has described each of the RAID levels available, how each level works, and what performance can be expected from each level. However, there is a tradeoff involved in fault tolerance; the tradeoff involves performance, fault tolerance, and economics. Table 15.5 compares the different RAID levels.

Table 15.5 Fault Tolerance Comparison

RAID Level	Performance	Fault Tolerance	Economics
RAID-0 (no fault tolerance)	Best	Worst	Best
RAID-1 (mirroring)	Good	Best	Worst
RAID-5 (data guarding)	Worst	Good	Good

As you can see, the choice is not easy. It is up to you to decide which type of fault tolerance is best for your particular requirements.

Summary

This chapter looked at how software and hardware disk arrays operate and how you can characterize the performance of disk arrays. The chapter also looked at how to tune the disk arrays for optimal performance by isolating the sequential I/Os and spreading out the random I/Os as much as possible.

The choice of which RAID level you use may not be an easy one. You must determine the number of I/Os per second that your system must support based on your application and number of users. Use this data and your budget to determine the type of fault tolerance that fits your needs.

Probably the most important topic discussed in this chapter was the difference between how you configure your system with a disk array and how you configure your system with traditional disks. When you use a disk array, you should abandon the configuration rules you have used in the past and put all your randomly accessed data files and indexes on the same logical volume. (This recommendation is the opposite of what you have heard in the past.)

Part III of this book, “Configuring the System,” looks at some specific configurations and examples of how disk arrays can be used in each of those configurations.

Part

Configuring the System

Chapter 16 OLTP System

- 17** Batch Processing System
- 18** Decision Support System
- 19** Data Warehousing System
- 20** BLOB System
- 21** The Oracle Parallel Server System
- 22** Optimal Backup and Recovery
- 23** Miscellaneous Configurations

Part II of this book explained how to tune the database server: the RDBMS itself as well as the server operating system. It also explored the computer hardware and touched on some of the ways you can configure the server itself to take advantage of specific data access patterns.

Part III of this book looks at specialized configurations based on those data patterns. The following chapters detail several different types of systems ranging from traditional OLTP and decision support systems to some of the newer data warehousing and BLOB systems. The chapters analyze the systems and determine the data access patterns of these types of applications. After characterizing the system, you look at some design considerations that most effectively take advantage of that knowledge. Finally, you will look at some of the ways the RDBMS and server operating system can be tuned to perform optimally in the particular type of configuration.

By understanding and characterizing the data access patterns on your system and understanding the characteristics of Oracle, your server OS, and hardware, you can effectively design an optimal system. Not all systems are alike and not all the things discussed in these chapters will apply directly to your system. Although each system is unique, the concepts will provide some useful insight about how you can optimize your own system.

Chapter

6

OLTP System

Probably the most common type of RDBMS system in use today is the OnLine Transaction Processing (OLTP) system. Almost everyone has some interaction with an OLTP system every day, whether they realize it or not. OLTP systems are used in grocery stores to keep track of items that have been sold and that must be restocked. OLTP systems are used in banks to keep track of savings and checking accounts. OLTP systems are used to keep track of how much money you charged to your credit card and to send you a bill each month. OLTP systems are used in the airline industry to keep track of schedules and seats sold on airplanes. In fact, every aspect of our lives is probably touched by some type of OLTP system.

This chapter looks at the OLTP system and analyzes the characteristics of the system and the data access patterns typically involved with the OLTP system. Having analyzed the system, you can then look at some of the ways you can optimally design and tune the system. Finally, you determine whether some enhancements can be made to the RDBMS, OS, or hardware to improve system performance.

Characteristics of the OLTP System

The OLTP system usually involves many users, each of whom accesses the system randomly by using various types of transactions. Although some OLTP systems use only one type of transaction at a time, most OLTP systems have many different types of transactions happening simultaneously. Whether one or many different types of transactions occur, the general characteristics of the OLTP system are the same:

- ◆ **Simultaneous access to data.** Large amounts of data is accessed simultaneously. These simultaneous accesses may or may not be to the same data in the system.
- ◆ **Data is accessed with both reads and writes.** In an OLTP system, typically all types of transactions occur. These can consist reads, writes, and deletions of data in the database.
- ◆ **The database grows.** Because OLTP systems have large numbers of inserts into tables, the size of the data tables and their associated indexes grow. This growth may be significant; you should design the system to accommodate this growth.
- ◆ **Large number of users.** The typical OLTP system must support many users simultaneously. The required number of users determines the type and size of the computer needed.
- ◆ **Response-time constraints.** Because the OLTP system services online users, certain response times must be achieved. Usually, response time is a critical component of the design of the OLTP system.
- ◆ **Continuous uptime.** An OLTP system is typically in operation 24 hours per day, 7 days a week without any downtime. Special considerations must be made for fault-tolerant components or backup systems.

Each of these characteristics requires special configuration considerations, as you see later in this chapter. The load and response time requirements, as well as the number of users to be supported, are all factors in the type and size of the computer used to meet those requirements.

Data Access Patterns

The data access patterns in a purely OLTP system are fairly straightforward. Based on the types of transactions you generate, you should be able to fairly accurately determine these patterns. Although each system has its own specific data access patterns, OLTP systems have the following general characteristics.

- ◆ Data access to an individual redo log file for logging is always sequential and write only.
- ◆ The LGWR writes to the redo log file in a size that may vary from the size of the redo entry up to the size of the redo log buffer.
- ◆ Data access to the individual redo log files for archiving is always sequential and read only.
- ◆ If more than one redo log file is on a disk volume and archiving is enabled (you should *always* archive), the access to those disks is not sequential because the disk heads must seek between the log writes and the archive reads.
- ◆ Data access to the data files is random. Data accesses to the data files may be random or sequential on a transactional basis but because many of these transactions occur simultaneously, the data files are always accessed in a random fashion. Most transactions are small.
- ◆ Most of the time, data reads and writes from the data files occur in DB_BLOCK_SIZE sizes. Because OLTP systems usually do not have very many table scans, it is unlikely that multiblock reads and writes will happen.
- ◆ The typical OLTP system is characterized by some data that is “hot” and some data that is “cold.” Hot data is accessed much more frequently than cold data. The general rule of thumb is that 80 percent of the data accesses occur in 20 percent of the data (a phenomenon sometimes known as the 80-20 rule).

These patterns vary depending on how your system operates, but the general principles are the same. You most likely store some historical data in a sequential manner. You probably store new data at the end of existing tables, thus limiting where the data is hot.

System Load

In an OLTP system, the system designers typically have a good idea what kind of load to expect on the system. These factors are used to determine the system configuration. A typical OLTP system must support a large number of users, which indicates that the system runs under a heavy load most of the time.

NOTE: In the real world, I typically see OLTP systems that show a significant I/O deficiency. By having an I/O capacity that does not fit the system load, you frequently see significant CPU idle cycles while you wait for I/Os to complete. An idle CPU can severely degrade performance.

The following list shows some of the typical load characteristics of an OLTP system:

- ◆ **Significant numbers of process switches or thread switches.** Because there are many user processes, the system must continually switch between these processes.

- ◆ **Heavy network traffic.** Each transaction most likely generates several network packets. Depending on the use of stored procedures, network traffic can be reduced.
- ◆ **Heavy I/O usage.** OLTP systems usually generate large numbers of random I/Os to the data files.
- ◆ **Heavy redo log usage.** Because OLTP transactions consist of both read and update activity, the redo log is heavily accessed.
- ◆ **Heavy use of rollback segments.** Because OLTP transactions consist of significant update activity, the rollback segments are heavily used.
- ◆ **Large amounts of memory.** The memory is used not only for the SGA, but for each of the server processes.

You can use these characteristics to help design and tune your OLTP system for optimal performance. The first step in the design process is to set goals for what you want to achieve.

Goals

The goal in tuning the OLTP system is to achieve a system with certain characteristics. Here are the characteristics of an optimally tuned OLTP system:

- ◆ **The system offers good response times.** For the majority of transactions, response times are within the specified range. Response time criteria usually specify a maximum response time for 90 to 95 percent of the transactions. (It is almost impossible to guarantee a certain response time for 100 percent of the transactions.)
- ◆ **The system does not show any characteristics of being drive bound.** Any disk bottleneck degrades performance. If the system is disk bound, you should either add more disks or increase memory.
- ◆ **Memory is sufficient.** If the machine is paging or swapping, performance is being severely degraded. The best alternative is to add more memory; if that is not possible, reduce the size of the SGA or the number of users until the system no longer pages or swaps.
- ◆ **There are a sufficient number of rollback segments to avoid rollback contention.** The system designer can easily avoid rollback contention by knowing the number of server processes and data access patterns.
- ◆ **There is sufficient space in the shared pool.** With large numbers of users, the shared SQL area is very important. Careful tuning is required to ensure that the shared pool isn't overloaded.
- ◆ **The system can handle peak loads.** It is not sufficient that the system can handle steady-state operations. During a checkpoint, the system must also provide adequate response times.
- ◆ **The system meets any additional requirements you have.** If you have requirements for continual archiving to a remote machine or a short window for backups, these factors must also be taken into consideration.

By setting goals for how you expect the system to perform, you can easily determine whether you are successful. You can also determine earlier in the design process whether you will be able to achieve those goals.

Review of OLTP Characteristics

By analyzing the characteristics of the OLTP system and the goals you want to achieve, you gather a lot of data with which you can build the system. Much of the design of the system is determined by the data access patterns and specifications of the system.

In an OLTP system, data access is generally random, with perhaps some sequential components. The OLTP system usually handles a lot of concurrent data access from many users simultaneously. By configuring the system and carefully distributing the I/O load, you can achieve a high level of concurrency.

I encourage you to relate the information in the next part of this chapter to your particular configuration. Look for similarities and differences and decide how you can benefit from the tuning guidelines presented in previous chapters. Remember that when we talk about the design of the system, we are talking about every aspect of the system, from the client application code to the hardware.

Many design considerations you use to set up your system are based on knowing your application and system. It is important to spend the time up front to carefully examine how the system should run; only then can you determine the design of the system. The end product depends enormously on the amount of time and effort you put into the original design. No amount of tuning after the system is in production can make up for poor design choices.

Design Considerations

The first part of this chapter provided you with a good idea of how the OLTP system operates based on the data access patterns. Before you look at the design of the system, here is a review of a few of the concepts described in earlier chapters:

- ◆ **I/Os are usually the limiting factor in your system.** You can only do a fixed number of random I/Os per second per disk drive (refer to Chapter 14, “Advanced Disk I/O Concepts”).
- ◆ **I/Os can be reduced by caching data blocks in the SGA.** If data you want to access is already in the SGA, a disk I/O is not required.
- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If seeks can be reduced, more I/Os per second can be achieved.

- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out among many drives, the overall rate can be increased.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

These factors contribute to determining the optimal data layout of the system. The physical layout—along with SGA and shared pool tuning—creates an optimally configured server for the typical OLTP application. Of course, proper design of the application is just as important, but that topic is addressed in Part IV of this book, “Tuning SQL.”

Physical Data Layout

This section looks at how the data on the system should be configured on the physical disks. First, it looks at how to lay out the data on traditional disks; then it looks at disk arrays. I recommend using disk arrays if at all possible; the ease of use and performance benefits are worth the cost of the array.

The main goal in designing the physical data layout is to isolate the sequential I/Os and to provide balanced I/Os across all the disks that are randomly accessed. Earlier in this chapter, you learned that the redo log files and the archive log files in an OLTP system are all accessed sequentially. You also know that the majority (if not all) of the data files are accessed in a random fashion.

Many of the concepts presented here represent the best-case scenario. I realize that budgetary constraints do not permit everyone to buy the optimal number of disks for this configuration. Make the most of these guidelines. Remember that it is the number of disks that provide performance in an OLTP system; if you have the opportunity to buy one 4 gigabyte disk drive or two 2 gigabyte disk drives, the best performance comes from the two 2 gigabyte disk drives.

Traditional Disks

The layout for the OLTP system is quite straightforward. A minimal configuration looks something like this:

Element (# of Disks)	Comments
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries.
Redo log (2+)	At the very least, the OLTP system should have two log disks so that you can mirror the logs by means of Oracle log groups. If you have only two log disks, performance is degraded during archiving because the sequential nature of the log writes is disrupted.

<i>Element (# of Disks)</i>	<i>Comments</i>
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary.
Data files (1+)	The number of disks you need for data is determined by the amount of random I/O your user community generates.
Index files (1+)	The number of disks needed for indexes is determined by the size of the indexes and the number of I/Os to the indexes.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

With traditional disk drives, you can stripe data among the various disks using either Oracle striping or OS striping. (I consider OS striping to be the same as using a disk array, as described in the following section.)

To stripe a table across multiple disks, you must first create the tablespace with multiple data files:

```
CREATE TABLESPACE striped_tablespace
  DATAFILE 'file1' SIZE 1M,
            'file2' SIZE 1M,
            'file3' SIZE 1M,
            'file4' SIZE 1M;
```

In this example, file1 is on disk 1, file2 is on disk 2, and so on. Once the tablespace is created, you cause the table to be striped when you create the table by making the number of extents equal to the number of data files:

```
CREATE TABLE striped_table
  ( column1 number(4),
    column2 varchar2(40) )
TABLESPACE striped_tablespace
  STORAGE ( INITIAL 512K NEXT 512K,
            MINEXTENTS 4 PCTINCREASE 0 );
```

In this example, the table is striped across all four of the data files. The problem with striping in this manner is that the size of the stripes is the size of the extent. Until all four extents have data in them, you access only one of the extents.

It can be difficult to manage the striping of hundreds of data files across hundreds of disks and to balance the load across these disks. If some tables are extremely hot (frequently accessed), you should stripe that table across many disk drives; cold tables (those infrequently accessed) can be striped across fewer disk drives.

It is the twin difficulty of configuration and load balancing that makes me prefer a disk array. When you use a disk array, these tasks are greatly simplified.

Disk Arrays

The layout for the OLTP system on a disk array is much simpler than it is on traditional disk drives. A minimal configuration looks something like this:

<i>Element (# of Volumes)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, this disk should be mirrored for additional fault tolerance.
Redo log (1+)	There should be at least one log volume. This volume should be made up of at least two physical disks using RAID 1. If you use only one log volume, performance is degraded during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you need to archive. This data can be written to tape as necessary.
Data and index files(1+)	Because you always get concurrent access to disks in a disk array, it is not necessary to split the data and indexes across separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

Unlike traditional disk drives, a disk array automatically stripes the data among all the disk drives. Therefore, you have to create only one tablespace and table for all your data. Although it is not necessary to put indexes into another tablespace, I recommend doing so for other reasons (such as monitoring and maintenance).

With traditional disk partitioning, it is difficult to manage hundreds of data files and hundreds of disks; with a disk array, on the other hand, it is possible to manage hundreds of disks with just a few data files. Of course, Oracle has a 2 gigabyte limitation on the size of a data file, but this is easily resolved by creating a data file for every 2 gigabytes of space you need. The data files can all reside on the same disk array volume.

The fact that some tables are hot and others are cold is irrelevant because all data access is uniformly distributed among all the disks. Load balancing is not an issue because the small size of the stripe ensures that random data accesses are most likely spread out among all the disks in the array.

By using a disk array, you greatly simplify many of the management and load balancing tasks. With the disk array, you also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

I recommend that you use a disk array if possible. Software striping is fine, but if your system is under heavy loads—as is typical in an OLTP system—you achieve better performance by offloading the striping overhead to a hardware RAID controller.

Hardware Considerations

When choosing hardware to use for an OLTP system, consider these several factors:

- ◆ **High user load.** Many concurrent processes/threads are simultaneously accessing the system.
- ◆ **High I/O load.** I/Os are concurrent and heavy with mostly random I/O.
- ◆ **High network traffic.**

Because many different processes use the machine at once, an SMP or MPP machine should scale very well. Because an SMP architecture uses CPUs based on the processes that are available to be run, if you always have a runnable process available for each CPU, you should see good scaling by adding additional processors. With an MPP machine, you see a similar effect but on a much larger scale.

Because of the many random accesses to the disks, you can benefit from a disk array. I prefer hardware striping to OS striping because hardware striping does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is a good alternative.

If network bandwidth is too high, consider adding additional network interface controllers (NICs) to try to reduce the bandwidth consumed on each segment. The use of stored procedures may also help reduce network traffic.

Tuning Considerations

The OLTP system is tuned to allow for many users to access the system and at the same to allow maximum throughput and minimal response times. All the tuning considerations presented in Part II of this book, “Tuning the Server,” apply here; there are no special tuning parameters to use in the OLTP system.

Both Oracle and the server operating system may require tuning. The following sections look first at Oracle and then at the server operating system.

Oracle Tuning

Analyze the following things carefully to determine whether adjustment to these parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** This is probably the most critical area in the OLTP system. Having an insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck. The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The ratio of PHYSICAL READS to DB BLOCK GETS and CONSISTENT GETS is the cache-miss ratio. This number should be minimized.
- ◆ **Library cache.** Remember to check The V\$LIBRARYCACHE table, which contains statistics on how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You should be able to reduce the library cache misses by increasing the amount of memory available for the library cache. You can do this by increasing the Oracle parameter SHARED_POOL_SIZE. Depending on the size of the shared pool, you may want to increase the cache by 10 percent and see how it performs.
- ◆ **Open cursors.** You may also have to increase the number of cursors available for a session by increasing the Oracle parameter OPEN_CURSORS. Set the number of OPEN_CURSORS based on the application. Remember that cursors are pointers to memory used by Oracle in the processing of transactions.
- ◆ **Cursor space for time.** If you have plenty of memory, you may be able to speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR SPACE FOR TIME to TRUE.
- ◆ **Spin counts.** By tuning the Oracle initialization parameter SPIN COUNT to enable the process to spin (instead of going to sleep) while waiting for an Oracle resource, you may see improved performance. SPIN COUNT is particularly useful when you have multiple CPUs and short-running transactions.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, check the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback contention.** Rollback contention occurs when too many transactions try to use the same rollback segment at the same time and some of them have to wait. You can tell if you have contention on rollback segments by examining the dynamic performance table V\$WAITSTAT. Check for an excessive number of UNDO HEADERs, UNDO BLOCKs, SYSTEM UNDO HEADERs, and SYSTEM UNDO BLOCKs. Compare these values to the total number of requests for data. If the number is high, you need more rollback segments.

- ◆ **Use many small rollback segments for OLTP.** Create many small rollback segments of perhaps 10K to 20K in size, with two to four extents each; perhaps you can have a rollback segment available for each server process.
- ◆ **Latch contention.** You can determine latch contention by examining the dynamic performance table V\$LATCH. Look for the ratio of MISSES to GETS, the number of SLEEPS, and the ratio of IMMEDIATE MISSES to IMMEDIATE GETS. If the miss ratio is high, reduce the size of `LOG_SMALL_ENTRY_MAX_SIZE` to minimize the time any one user process holds the latch; alternatively, increase the value of `LOG_SIMULTANEOUS_COPIES` to reduce contention by adding more redo copy latches. If neither of these parameters help, set the initialization parameter `LOG_ENTRY_PREBUILD_THRESHOLD`. Any redo entry of a size smaller than the size you set this parameter must be prebuilt. This reduces the time the latch is held.
- ◆ **Checkpoints.** It may be necessary to tune checkpoints under certain circumstances. Although this is usually not necessary, if you see severely degraded performance during checkpoints, you can reduce the effect by enabling the CKPT process. Do so by setting the Oracle initialization parameter `CHECKPOINT_PROCESS` to TRUE.
- ◆ **Archiving.** By now, you should be convinced that you should always run with archiving enabled. By adjusting the initialization parameters `LOG_ARCHIVE_BUFFERS` and `LOG_ARCHIVE_BUFFER_SIZE`, you can either slow down or speed up the performance of archiving. By speeding up archiving, the effect on the system is of a shorter duration but is more noticeable. Slowing down archiving lengthens the duration but reduces the effect.

These are some of the areas to which you should pay particular attention when tuning a system used primarily for OLTP. The areas that probably require the most attention are I/O and memory because they are so closely related. By optimizing the use of memory, you may be able to reduce I/Os, which are probably running near the hardware limitations.

Server OS Tuning

You may have to tune the server OS to facilitate the large number of processes and network connections required of an OLTP system. The server OS also affects the optimization of I/O performance. Following are some of the things you may have to tune in the server OS; keep in mind that some OSes may not require any tuning in these areas:

- ◆ **Processes.** The system must be tuned to handle large numbers of processes.
- ◆ **Memory.** The system should be tuned to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes.
- ◆ **Memory enhancements.** Take advantage of 4M pages and ISM if they are available. Both of these features improve Oracle performance in an OLTP environment.

- ◆ **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- ◆ **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In an OLTP environment, these features simply waste CPU cycles.
- ◆ **Cache affinity.** If possible, turn off cache affinity. In a heavily used OLTP environment, cache affinity does not benefit you; it only wastes CPU cycles.
- ◆ **Network.** Tune the system to favor your network protocol. Also configure the network for optimal Oracle performance. You may want to change the network packet size.

The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, you can increase Oracle performance. This topic is discussed in more detail in Chapter 12, “Operating System-Specific Tuning.”

Enhancements

You may want to make some additional enhancements to improve the performance of the OLTP system. Listed here are some of the items that can offer performance improvement; also listed are a few items that do not improve performance. Each item is accompanied by an explanation.

- ◆ **Block size.** Because typical OLTP transactions are small in nature and many of them happen simultaneously, it is not beneficial to increase the database block size. The default value of 2K is probably sufficient.
- ◆ **Clusters.** Depending on the application, you may benefit from clusters. The benefit you receive depends exclusively on the data access patterns and your application.
- ◆ **Hash clusters.** OLTP systems typically access tables with some sort of ID, such as a part number or account ID. If this is the case in your application, you may enhance performance by creating hash clusters on some of those tables.
- ◆ **Indexes.** Based on how the data is accessed, you may see significant performance benefits by using indexes. How and when to use indexes is described in Chapter 9, “Oracle Instance Tuning.”
- ◆ **Multiblock reads and writes.** In the OLTP environment, it is very unlikely that you will benefit from multiblock reads and writes because very little data access involves sequential access to data.
- ◆ **Parallel query.** The Oracle Parallel Query option does not offer any benefit in the OLTP environment. Parallelizing queries offers the benefit of making long-running queries concurrent. Because most OLTP transactions are small and of short duration, there is not much of a benefit.

Many of these enhancements can help your performance. The specific effect on your system varies, but the information presented here is true for most systems.

Oracle Parallel Server Option

The Oracle Parallel Server option may be very beneficial to your operation in two ways:

1. **Performance.** If your system is a candidate for the Parallel Server option, you should see significant performance improvements.
2. **Fault tolerance.** Many OLTP installations demand no downtime. By using the Oracle Parallel Server option, you can keep the system running, even if one computer fails.

The Oracle Parallel Server option can enhance the performance of your OLTP system—but only if the system is suitable for the Parallel Server option. To see scaleable performance increases, the system must meet the following criteria:

- ◆ **Partitionable data.** The data should be partitionable. In other words, the work related to certain tables should be done on one computer while work related to other tables should be done on another computer. If all users access the exact same data, your application may not be a good candidate for the Parallel Server option. Although some overlap is fine, the majority of the data should be partitionable.
- ◆ **Many processes.** If you have only a few users and concurrent transactions, your system is probably not a good candidate for the Parallel Server option.

If these criteria are met, you can probably benefit from using the Oracle Parallel Server option.

NOTE: The Oracle Parallel Server option is not available on all platforms. Consult your system provider or Oracle to determine which platforms support the Oracle Parallel Server option.

Hardware Enhancements

In an OLTP environment, you can make several hardware enhancements to help improve performance. These hardware enhancements can be beneficial in the areas of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements—assuming that you are not I/O bound. Computer manufacturers are continually making improvements to system processors, not only in speed of the CPU itself but in improved CPU caches as well.

If you already have an SMP or MPP machine, enhancing the CPU may be as easy as adding another processor. Before you purchase an additional processor of the same type and speed, however, see whether *upgrading* to a faster processor can provide more benefit than *adding* a CPU. For example, upgrading from a 66 MHz processor to a 133 MHz processor may provide more benefit than purchasing an additional 66 MHz CPU—with the added benefit that you now have the option of adding more 133 MHz CPUs. How you enhance your CPU will depend on the upgradability of your computer as well as your budget.

SMP and MPP computers provide scalable CPU performance enhancements at a fraction of the cost of another computer. When upgrading your processors or adding additional processors, remember that your I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when you add processors.

I/O Enhancements

You can enhance I/O by adding disk drives or purchasing a hardware disk array. OLTP really benefits from disk striping as done by software or hardware disk arrays. If you use Oracle data file striping, the size of the data stripe is so large that there is some dependency on the location of the data being striped. If the data currently being accessed is all in one extent, the effect of Oracle striping is minimal.

By using hardware or software striping with a small stripe size (for example, 16K), you see a natural balancing of I/Os across all the disk drives. This striping provides better load balancing than is possible when you try to balance the load across disks by hand.

Hardware and software disk arrays also have the benefit of optional fault tolerance. As you saw in Chapter 15, “Disk Arrays,” each of the fault tolerant RAID levels has its advantages and disadvantages. You must first choose the correct fault tolerance for your needs and then make sure that you have sufficient I/O capabilities to achieve the required performance level. If you use fault tolerance, you most likely have to increase the number of disk drives in your system.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk and to mask the overhead associated with fault tolerance. If you use RAID-5, you see minimal performance degradation if the controller has a write cache (as long as you don’t exceed the performance of the cache and disks). The write returns to the OS as soon as the data has been written to the write cache.

CAUTION: It is important to make sure that any controller you use with a write cache is protected against a power failure. Some of the disk array controllers on the market today offer a battery-backed, mirrored memory cache to protect your data. Remember: Once Oracle believes that data has been written to the drive, that data had better be there.

Enhancements to the I/O subsystem almost always help in an OLTP system because most OLTP systems are disk bound. Be sure that you have a sufficient number of disk drives, properly configured. I/O is probably the most important area of OLTP server tuning. Of course, the most effective way to get optimal performance from the system is to have both a well-tuned server and a well-tuned application.

Network Enhancements

If the network is a bottleneck in your system, you can improve performance by adding a faster network interface controller (NIC) or by segmenting the network into pieces and accessing the server through multiple networks.

In recent months, 100 megabit-per-second Ethernet has emerged as the standard for Ethernet networking. If you upgrade existing 10 megabit-per-second Ethernet, your performance improvement can be enormous. Even if you have 100 megabit-per-second Ethernet, you may find it useful to segment the network into several subnets. Both approaches allow more network packets through to the server. If you find that your network is used more than 60 to 70 percent, you should seriously think about upgrading.

Depending on your configuration, you may benefit from the use of Token Ring or fiber-optics networking. Frequently, the network hardware you already have dictates the route you take in terms of new network hardware.

Miscellaneous Enhancements

You may be able to take advantage of other hardware or software enhancements to improve OLTP performance. Some of these performance enhancements are very innovative. The following sections look at a few of these enhancements.

Etherplex from Systek

The Etherplex board from Systek provides hardware login support for UNIX systems. The board includes an Ethernet controller and provides TCP/IP support as well as rlogin and telnet support. The Etherplex board uses a high-speed RISC processor to do its work and provides support for up to 2,048 network connections.

By offloading the terminal and network overhead to the Etherplex board, you can dedicate the system's CPUs and memory to the application and to Oracle. Because the Etherplex board provides only rlogin and telnet support, it cannot help in a client/server environment, but if your application uses terminal logins, Etherplex can greatly enhance your system.

Transaction Monitors

Transaction Monitors can perform many functions: from multiplexing Oracle connections to providing system fault tolerance. By using a Transaction Monitor (TM), you can offload much of the work performed by Oracle to one or more front-end machines.

Because much of the system memory is used by the server processes, using a TM can reduce the number of server processes on the server by offloading them to the front-end machines. Here's how TM works.

Transaction Monitors provide a mechanism to use in your application code that allows you to queue requests to an Oracle back-end process. The front-end of the TM takes Oracle requests and places them in a queue. The back-end of the TM takes the requests off the queue and transmits them to Oracle through SQL*Net.

Although there may be hundreds or thousands of front-end processes, you can limit the back-end processes connected to Oracle. Each user process creates a front-end process; the number of back-end processes is determined by the database administrator or applications developer.

By using a TM, you can support hundreds or thousands of users and at the same time limit the number of connections into the database. You can use a TM to save server memory or to reduce contention. By separating the queues into different types of transactions, you can limit certain transactions (thus throttling them).

NOTE: If you have exceeded the limits of your system and have no other way of reducing the load, taking advantage of a product such as Etherplex or Transaction Monitor may be the solution for you. You sometimes have to be innovative to fulfill your performance goals.

Performance Verification

Once you design and build your system, you must have some way of verifying that you can achieve the required throughput and user load. Verification can be extremely difficult, time consuming, and expensive—especially in a client/server environment. A class of very good load-testing software and hardware is available that can simulate both local and network users, but it is very expensive.

Writing your own code may be the best alternative for load testing. You should consider several things when building load-testing software:

- ◆ **Simulate many processes.** A load test that provides for a heavy load but uses only a few processes does not properly simulate an OLTP load. Some areas of the system may go untested if you do not create many processes.
- ◆ **Use the network.** If your production users will connect through the network, use the network for the load test. Make sure that you use the same network protocol.
- ◆ **Simulate the transaction.** Try to make the test transactions simulate the actual transactions as closely as possible. The closer you get in the simulation, the better the test.
- ◆ **Put monitoring into the test.** By measuring response times, you can get a good idea of how the system will perform under load.

By putting together a good test program, you not only test the system for bugs, you can get a good idea about whether tuning changes are helping or hurting. The following sections look at what you should test in the RDBMS and the operating system.

What To Test in the RDBMS

You should test the RDBMS with functionality as well as load testing and performance in mind. Make sure that the system will function as specified with the required number of user connections. Usually, these connections do not have to be real users for the test, but you should make sure that you can invoke the application the same number of times as the required user load.

Many times, applications function properly with a few users connected but when a large number of users is connected, contention or other problems bring the system to a halt. You must test both the functionality and the performance of the application and the RDBMS.

Once you build a test load that stresses the system by simulating a large number of connected users, you can start looking at performance. Check both response times and throughput to verify that you can handle the required load. If you design a test that simulates the maximum number of users the system is required to handle, be sure to put simulated delays (for keying-in time and think time) into the test.

Real-world users vary in the amount of time they take to input data into the system because most input requires some sort of interaction. Once the data is displayed on the display device, users vary in the amount of time it takes them to view the data and think about it. Program some variation into the keying-in and think times so that all the users don't work on exactly the same cycle.

Once you build an effective load test with repeatable performance results, start changing some of the parameters described in Part II of this book and that you have seen in this chapter. Change only one parameter at a time so that you can determine which changes affected performance and by how much.

Before changing anything, define your expectations for the changes you are going to make. If the results don't turn out as expected, try to understand why they didn't. Be sure that you log your results so that you have a reference of what changes were effective and what changes were not.

What To Test in the OS

For the most part, the OS is tested with the RDBMS. If you want to change some specific OS parameter that is needed to increase a limitation, you usually don't have to retest the performance just for the OS change. Any limit change is usually associated with an RDBMS change, and the two can be tested together.

Other changes such as feature enhancements should also be tested with the RDBMS, but you should watch and analyze the results more closely. Things such as scheduler changes and cache

affinity that may have an adverse effect should be analyzed very closely to verify that performance has not degraded.

The OS is primarily a vehicle for the RDBMS to use. The primary goal in tuning the server OS is to reduce overhead and optimize the I/O, memory, and networking subsystems.

Benchmarks

Standardized benchmarks are a good way to judge how well a system is performing and to compare different hardware platforms. If they are available to you, standardized benchmarks are also a good way to analyze the performance of your particular platform (refer to Chapter 5, “Benchmarking”).

Using a standardized benchmark such as the TPC-C benchmark is not a good way to tune your system. Each system should be tuned individually based on its own characteristics. A system that has been tuned for a benchmark, such as TPC-C, can provide useful tuning hints. Because the primary goal of a TPC benchmark is for the sponsor to get the best possible performance, you can see how the sponsors have optimally tuned the system.

Because an official TPC-C benchmark that uses Oracle is usually submitted by the hardware vendor, OS vendor, and Oracle, you can be assured that the system has been tuned as optimally as possible. If you get the chance, try to obtain a Full Disclosure Report from the TPC (on the Web at <http://www.tpc.org>) and look at the way the system was designed and tuned.

Of course, the best benchmark for your system is one designed and built to run your application. This benchmark provides you with all the information you need to judge the amount and size of the hardware required. Your custom benchmark will also provide you with valuable information about any configuration changes you make.

The design and function of your benchmark is based entirely on your application and system configuration. Try to create a benchmark that has a variable load so that you can push the system to its limits to gather maximum load information.

Summary

OLTP systems are probably the most common RDBMS systems in use today. As such, a wealth of tuning knowledge is available. This chapter examined the following characteristics of the OLTP system:

- ◆ **Significant numbers of user processes.** OLTP systems usually support hundreds or thousands of users.
- ◆ **Heavy I/O usage.** Each user concurrently accesses many different tables with both read and update activity, causing many concurrent I/Os on the system.

- ◆ **A strict response time criteria.** Strict response times are characteristic of the OLTP system. The OLTP system supports online users, and online users demand reasonable response times. Each second the user waits may cost the company money.
- ◆ **A strict uptime criteria.** In most OLTP configurations, downtime is not an option. The system may have to be up 24 hours a day, 7 days a week.

This chapter analyzed the characteristics of the system and analyzed the data access patterns typically involved with the OLTP system. The chapter then looked at some of the ways the system can be optimally designed and tuned. You also learned about some enhancements you can make to the RDBMS, OS, and hardware to improve system performance (you also learned about some enhancements that don't help). The last section of the chapter examined the verification methods you can use to confirm that your tuning efforts have positively affected the performance of the system.

The next chapters look at some of the other classes of systems with which Oracle is used, including decision support systems, batch processing systems, and others.

Chapter

7

Batch Processing System

Batch processing systems are different from OLTP systems in that batch jobs are not run interactively. Batch jobs are submitted at some later time (perhaps much later) and run until they are done. The batch processing system typically contains many of the attributes of both the OLTP system and the decision support system, with some additional unique attributes. It is difficult to characterize a batch processing system accurately because the types of jobs run in batch tend to vary quite a lot.

To generalize, batch processing usually involves loading and unloading large amounts of data on the system, creating indexes, detail reporting, and so on. Batch processing often occurs when no other work is being done on the system.

The types of work typically done in batch may include, but are not limited to, some of the following activities:

- ◆ Summary and detailed reports
- ◆ End-of-month, end-of-quarter, and end-of-year books
- ◆ Database imports/exports
- ◆ Payroll processing
- ◆ Database merges

These are just a few of the types of activities typically run in batch. The following section looks at some of the characteristics of batch processing.

Characteristics of the Batch Processing System

Although there is no such thing as a “typical” batch processing system, there are some characteristics of batch processing. A batch processing system usually has some of the following characteristics:

- ◆ **The data to be processed on the system is loaded just before processing.** In fact, the loading of the data is considered part of the batch processing job.
- ◆ **Index creation.** The data is often indexed immediately after loading and before processing.
- ◆ **The types of processing varies.** Processing can consist of joins, sorts, and so on; anything is possible based on the processing job.
- ◆ **Queries are much more complex in a batch processing job than they are in OLTP jobs.**
- ◆ **Generates intense activity on the part of the server components.** Most components of the system run at 100 percent for the duration of the job.

The load on the batch processing system is very high for a certain length of time. In fact, because response times are not an issue for batch processing, it is reasonable to push the system to its limits. Rather than spreading out the load and trying to achieve reasonable response times (as you do in an OLTP system), the goal of the batch processing system is to push the system to its limits to get the work done in as short an amount of time as possible.

It is not uncommon for batch processing systems to have fairly strict windows in which to do their job. It is therefore essential that the system run at 100 percent until that job is completed.

The batch processing system has several unique characteristics:

- ◆ **Batch processing systems are typically dedicated to the job at hand.** No other processing occurs on the system.
- ◆ **Database loads and unloads are typically part of batch processing.**
- ◆ **Index creation is typically part of the batch processing job.**

These characteristics are unique to batch processing systems and can tell us quite a bit about the data access patterns we should expect to see in this type of system.

Data Access Patterns

Because of the unique characteristics of the batch processing system, you will probably see the following data access patterns on this type of system:

- ◆ **Heavy I/O usage.** Batch processing systems usually generate large amounts of I/O to the data files. During load and unload, this activity is sequential. The type of I/O during the query processing depends on the type of processing being done.
- ◆ **Heavy sequential I/O to load/unload files.** The data being loaded creates heavy sequential I/O, as does the unloading of the results.
- ◆ **Heavy index file access.** During index creation, the indexes are used very heavily.
- ◆ **Moderate to heavy redo log usage.** The amount of access to the redo log depends on the type of processing being done.
- ◆ **Moderate to heavy use of rollback segments.** The amount of access to the rollback segments also depends on the type of processing.
- ◆ **Large amounts of memory are used.** The memory is used not only for the SGA but for sorting and so on.

You can use these characteristics to help design and tune the batch processing system for optimal performance. The first step in this design process is to set goals for what you want to achieve.

System Load

In the batch processing system, you normally try to run the system at 100 percent during the batch job. In contrast, in an OLTP system, you hold some system capacity in reserve for peak times or checkpoints. Unlike OLTP systems, where there are many users with small queries, a batch processing system typically has only one job running on the system. This job should take advantage of the entire capacity of the system. By avoiding I/O bottlenecks, this is achievable.

NOTE: Usually, I see systems configured with an I/O capacity not suitable for their required tasks. By having an I/O capacity insufficient for your work load, you will most likely see a significant I/O bottleneck, which can severely degrade performance.

The following list shows some of the typical load characteristics of a batch processing system:

- ◆ **A small number of processes on the system.** This is true unless you are taking advantage of the Parallel Query option, which adds more processes and subsequently more process switches.
- ◆ **Minimal network traffic during transaction processing and significant network traffic during load and unload.** Typically, there is very little network traffic during the time the batch transactions are running. If you use a LAN to load and unload the database, there is significant network traffic during those times.
- ◆ **Heavy I/O usage.** Batch processing systems usually generate large amounts of I/O to the data files. These I/Os may be somewhat random during the transaction processing phase of the job but are sequential during the load and index creation phases. If you can take advantage of the Oracle Parallel Query option, you will see more random access to the index files.
- ◆ **Moderate to heavy redo log usage.** The amount of load on the system caused by use of the redo log depends on the type of transactions you are running.
- ◆ **Moderate to heavy use of rollback segments.** The amount of load on the system caused by use of rollback segments depends on your transactions.
- ◆ **Moderate to large amounts of memory.** In some batch processing jobs, a large amount of memory is used; in other jobs, only a small amount of memory is necessary. Things like large sorts and joins can take advantage of large amounts of memory. Exports and direct loads do not need significant amounts of RAM.

By looking at the characteristics of a batch processing system, you can make decisions about how some design ideas can help performance. But before you look at some design ideas, you should first set the goals you want to achieve from the design.

Goals

The goal in tuning a batch processing system is to build a system that has certain characteristics. Here are the characteristics of an optimally tuned batch processing system:

- ◆ **The system should be CPU bound.** When you remove all other bottlenecks, the system should be able to process at its limits: the speed of the CPUs.
- ◆ **The system is not disk bound.** Any disk bottleneck degrades performance. If this is the case, you should add more or faster disks.
- ◆ **Memory is sufficient.** If the machine is paging or swapping, performance is being severely degraded. The best alternative is to add more memory; if that is not possible, reduce the size of the SGA or the number of users until the system no longer pages or swaps.
- ◆ **The system meets any additional requirements you have.** With many batch processing systems, the work must be completed within a certain time. You may have this or some other constraint on your system.

Goal setting is always a good idea. By setting goals, you can judge the success or failure of the system in a quantitative manner. Goal setting can also help you properly size the system necessary for the job.

Review of Batch Processing Characteristics

By characterizing the system and examining the various aspects of your particular workload, you can design a system that takes advantage of your system's particular qualities. Each system has its own characteristics and workload and should be designed to take advantage of them.

Making generalizations about a batch processing system is practically impossible because of the different types of jobs done in batch. The processes your particular system does should determine your particular configuration.

Your specific batch job may involve only query processing or may involve large numbers of updates. You may or may not do large joins or sorts. *Batch* is such a general term that it is impossible to make specific assumptions about what you will be doing.

Here are the particular assumptions this chapter makes:

- ◆ Database loads and unloads are a part of the batch processing job. Loads may occur through import or using the database loader. The term *unload* refers to exports or backups or some other method of extracting the data from the database.
- ◆ Index creation is part of the batch processing job. After the data has been loaded, indexes are created.
- ◆ Only one job is running on the machine at a time. No concurrent jobs are being done. This is not to say that the batch job does not perform various functions; rather, the concurrency of transactions can be determined by the database administrator and application designer.

Because batch systems have such diverse functions, I recommend that you examine the design suggestions in the next section and see how they may or may not relate to your system configuration. By doing so, you can determine which ideas will work for you and which will not.

Your system design should be based on the unique characteristics of your application. Remember that the more time you spend up-front in database and application design, the more optimized your system will be. The combination of well-tuned hardware, OS, and RDBMS contributes to the performance of the system as a whole.

Design Considerations

Data access patterns give you the most information about how to design the optimal system. Here are some of the I/O characteristics with which you are familiar from previous chapters:

- ◆ **I/Os are typically the limiting factor in the system.** The number of I/Os that can happen is determined by the speed and number of disk drives (see Chapter 14, “Advanced Disk I/O Concepts”).
- ◆ **I/O can be reduced by caching data blocks in the SGA.** If the data you want to access is already in the SGA, a physical disk I/O is not required.
- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is. If you can reduce seek times, you can achieve more I/Os per second. In a batch processing system, the load/unload files are all accessed sequentially.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os over many drives, you can increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

Many or all of these factors may apply in your batch processing system, depending on the type of batch processing you do. The physical layout as well as SGA and shared-pool tuning can create an optimally configured server for these jobs. As you are aware, the design of the transactions is an integral part of the design of the entire system. The application, the RDBMS, the OS, and the hardware all work together to do the work. Neglecting any of these components can result in performance degradation.

Physical Data Layout

This section examines how the data should be configured on the physical disks for optimal performance in a batch processing environment. First, it looks at how to lay out the data on traditional disks; then it looks at disk arrays. I recommend using disk arrays if at all possible; the ease of use and performance benefits make disk arrays an investment worth making.

The goal in designing the physical data layout is to isolate the sequential I/Os and to balance I/Os across all the disks that are randomly accessed. Because batch processing systems may or may not do significant numbers of updates, you should weigh the options presented in the following sections against your particular configuration.

The recommendations that follow are based on light, moderate, and heavy use of the redo log files (as determined by the amount and intensity of updates to the system). As always, budgetary constraints may not allow everyone to buy the optimal number of disks for the recommended

configuration. Make the best of these guidelines. Remember that it is the number of disks that provide performance with random I/O; if you have the opportunity to buy one 4 gigabyte disk drive or two 2 gigabyte disk drives, the best performance comes from the two 2 gigabyte disk drives.

Traditional Disks

The optimal layout for a batch processing system is hemmed in by significant number of “it depends.” Read these guidelines, use what will work for you, and discard the rest. A minimal configuration should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries.
Redo log (0)	<i>Light update activity.</i> If there is little update activity, the redo log files do not see much I/O and are not a bottleneck. The redo log files can reside on the same disk as the system to save space; reallocate the disks for data files.
Redo log (2+)	<i>Moderate to heavy update activity.</i> If there is moderate to heavy update activity, it is much more important to protect the redo logs and provide for some sort of fault tolerance. The number of disks you need depends on the extent of the updates.
Archive logs (0)	<i>Light update activity.</i> As with the redo log files, archiving is minimal and not necessary in systems with little update activity. The archive log files can reside on the system disk.
Archive logs (1+)	<i>Moderate to heavy update activity.</i> As with redo log files, if updates are done on this system, you must provide for the sequential nature of the archive log files.
Data files (1+)	The number of disks you need for data is determined by the amount of random I/O your user community generates.
Index files (1+)	The number of disks needed for indexes is determined by the size of the indexes and the number of I/Os to the indexes.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can push a disk drive only to a maximum random I/O rate.

In earlier chapters, you learned that the data and indexes can be striped across the disks using Oracle or RAID striping or a combination of the two. I recommend a hardware disk array over manual Oracle striping primarily because the disk array is easier to use and has better performance. When you use a disk array, the task of distributing I/Os is greatly simplified.

Disk Arrays

As was true when you tried to lay out a batch processing system on traditional disks, the optimal layout is constrained by number of “it depends.” Read these guidelines, use what will work for you, and discard the rest. A minimal configuration should look something like this:

<i>Element (# of Volumes)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance.
Redo log (0)	<i>Light update activity.</i> If there is little update activity, the redo log files do not see very much I/O and are not a bottleneck. The redo log files can reside on the same disk as the system to save space; reallocate the disks for data files (if you do this, I recommend mirroring the OS volume).
Redo log (1+)	<i>Moderate to heavy update activity.</i> If there is moderate to heavy update activity, it is much more important to protect the redo logs and provide for some sort of fault tolerance. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance degrades during archiving because the sequential nature of the log writes is disrupted. But with a batch system, if your redo log is large enough, archiving can be deferred until after the batch transactions are complete.
Archive logs (0)	<i>Light update activity.</i> As with the redo log files, archiving is minimal and not necessary. The archive log files can also reside on the system disk.
Archive logs (1+)	<i>Moderate to heavy update activity.</i> As with the redo log files, if updates are done on this system, you must provide for the sequential nature of the archive log files. This data can be written to tape as necessary.
Data and index files (1+)	<i>Parallel Query option.</i> The number of disks used for data is determined by the amount of random I/Os generated by the transactions. If you use the Parallel Query option, you should share the drives between index and data.

<i>Element (# of Volumes)</i>	<i>Comments</i>
Data files (1+)	The number of disks you need for data is determined by the amount of random I/O your user community generates.
Index files (1+)	The number of disks needed for indexes is determined by the size of the indexes and the number of I/Os to the indexes.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can push a disk drive only to a maximum random I/O rate.

NOTE: Batch processing systems differ from both OLTP and decision support systems in that I recommend separating the index and data files onto different logical striped volumes. I make this recommendation because of the time constraints and frequency of index creations. If you use the Parallel Index Creation feature in the Oracle Parallel Query option, these I/Os are randomized; you are better off sharing data and index volumes to maximize the number of disks and spread out random I/O. I also recommend that you use the Parallel Query option.

Unlike traditional disk drives, a disk array automatically stripes the data among all the disk drives. Therefore, you have to create only one tablespace and table for all your data. Although it is not necessary to put indexes into another tablespace, I recommend doing so for other reasons (such as monitoring and maintenance).

With traditional disk partitioning, it is difficult to manage hundreds of data files and hundreds of disks; with a disk array, on the other hand, it is possible to manage hundreds of disks with just a few data files. Of course, Oracle has a 2-gigabyte limitation on the size of a data file, but this is easily resolved by creating a data file for every 2 gigabytes of space you need. The data files can all reside on the same disk array volume. By splitting tablespaces into several data files with tables striped across them (even if they reside on the same logical volume), you are also better able to take advantage of the Parallel Query option.

When you use a disk array, many of the management and load-balancing tasks are greatly simplified. You also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

I recommend that you use a disk array if possible. Software striping is fine, but if your system is under heavy loads—as it is in most batch processing activities—you are better off offloading that overhead to a hardware RAID controller.

Hardware Considerations

Consider these factors when choosing hardware to use for batch processing:

- ◆ **Low user load.** Not very many concurrent processes/threads simultaneously access the system (unless you take advantage of the Parallel Query option).
- ◆ **High I/O load.** I/Os may be a mixture of random and sequential, depending on the type of transactions.
- ◆ **Low network traffic during transaction processing, high network load during data load/unload.**

If you can take advantage of the Oracle Parallel Query option, many different processes will use the machine at once; an SMP or MPP machine should scale very well. Because an SMP architecture uses CPUs based on the processes that are available to be run, if you always have a runnable process available for each CPU, you should see good scaling by adding additional processors. With an MPP machine, you see a similar effect but on a much larger scale.

If you have many concurrent accesses to the disks, you benefit from a disk array. I prefer hardware striping to OS striping because hardware striping does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is a good alternative.

During transaction processing, there is no significant network traffic, but during the load and unload processes, you may see significant network activity. If you use a high-speed network interface such as 100Base-T or fiber optics, you will see increased throughput during the load and unload processes.

Tuning Considerations

The batch processing system is tuned for a minimal number of jobs to run at maximum throughput. The system has little concern for response times and much concern for throughput. All the tuning tips described in Part II of this book, “Tuning the Server,” apply here—with an emphasis on throughput.

Both Oracle and the server operating system may have to be tuned. The following sections look first at Oracle and then at the server operating system.

Oracle Tuning

Analyze these things carefully to determine whether adjustment to these parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** Block buffers are not nearly as critical in a batch system as they are in the OLTP system. Still, having an insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck. The statistics for the

buffer cache are kept in the dynamic performance table V\$SYSSTAT. The ratio of PHYSICAL READS to DB BLOCK GETS and CONSISTENT GETS is the cache-miss ratio, which should be minimized.

- ◆ **Library cache.** Check the V\$LIBRARYCACHE table that contains statistics about how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You should be able to reduce the number of library cache misses by increasing the amount of memory available for the library cache. Do this by increasing the Oracle tunable parameter SHARED_POOL_SIZE. Increase the size of the shared pool by 10 percent and run again. If that is not sufficient, increase the shared pool by another 10 percent until you are satisfied with the cache-hit rate.
- ◆ **Multiblock reads.** Because many of the queries involve table scans, it is important to make sure that you can take advantage of multiblock reads. The number of blocks read in a multiblock read is specified by the Oracle initialization parameter DB_FILE_MULTIBLOCK_READ_COUNT. Multiply this value by the value in the DB_BLOCK_SIZE parameter to obtain the maximum size of the I/Os. A good value for these I/Os is 64K.
- ◆ **Cursor space for time.** If you have plenty of memory, you may be able to speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR_SPACE_FOR_TIME to TRUE.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, check the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback segments.** Depending on the number of updates, you can reduce contention by increasing the number of rollback segments. Unlike an OLTP system, a batch processing system is better off with more rollback segments of a larger size. The size of the rollback segments depends on the size of the transactions. Rollback contention occurs when too many transactions try to use the same rollback segment at the same time and some of them have to wait. You can tell if there is contention on rollback segments by looking at the dynamic performance table V\$WAITSTAT. Check for an excessive number of UNDO HEADERs, UNDO BLOCKs, SYSTEM UNDO HEADERs, and SYSTEM UNDO BLOCKs. Compare these to the total number of requests for data. If the number is high, you need more rollback segments.
- ◆ **Latch contention.** Latch contention can be determined by examining the dynamic performance table V\$LATCH. Look for the ratio of MISSES to GETS, the number of SLEEPS, and the ratio of IMMEDIATE_MISSES to IMMEDIATE_GETS. If the miss ratio is high, reduce the size of LOG_SMALL_ENTRY_MAX_SIZE to minimize the time any user process holds the latch; alternatively, increase the value of LOG_SIMULTANEOUS_COPIES to reduce contention by adding more redo copy latches. If

neither of these parameters helps, set the initialization parameter `LOG_ENTRY_PREBUILD_THRESHOLD`. Any redo entry of a size smaller than the size you set this parameter must be prebuilt. This reduces the time the latch is held.

- ◆ **Checkpoints.** Under certain circumstances, you may have to tune checkpoints. Although this is usually not necessary, if you see severely degraded performance during checkpoints, you can reduce this effect by enabling the CKPT process. Do so by setting the Oracle initialization parameter `CHECKPOINT_PROCESS` to TRUE. With a batch processing system, you can usually build the redo log files large enough to defer the checkpoint until after transaction processing is finished.
- ◆ **Archiving.** As with the checkpoint process, you can defer archiving until after transaction processing is complete.

You should pay particular attention to these areas when tuning a system for batch processing. With batch processing, you have to tune not only for optimal transaction processing but for quick load and unload as well. Optimization of the index creation process can also help overall batch performance.

Server OS Tuning

You may have to tune the server OS to provide for a large number of processes and network connections; the server OS also affects the optimization of I/O performance. Following are some of the things you may have to tune in the server OS; keep in mind that some OSes may not require any tuning in these areas:

- ◆ **Memory.** The system should be tuned to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes. You may need significant amounts of memory for sorts and joins.
- ◆ **Memory enhancements.** You should take advantage of both 4M pages and ISM if they are available. Both features improve Oracle performance in an batch environment.
- ◆ **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- ◆ **Network performance.** Because loading and unloading of data is so critical to the overall performance of the batch jobs, you can increase network throughput by adding a high-speed network interface.
- ◆ **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In a batch system, allowing a process to run to completion (that is, not preempting it) is beneficial.
- ◆ **Cache affinity.** With a batch processing system, you may see some benefits from cache affinity. Because the processes tend to run somewhat longer, you may see some benefits.

The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, you can enhance Oracle performance.

Enhancements

Some additional enhancements may help improve the performance of your batch processing system. A few of the items that can help are listed here; the list also includes some enhancements that will not help performance. Each item is accompanied by an explanation.

- ◆ **Block size.** In a batch processing system, you may find that increasing the database block size can improve performance and reduce the amount of wasted space in the database. In a typical batch processing environment, many of the queries access the database in a sequential manner; a larger block brings more of these rows into the SGA at once. Having these additional rows in the SGA increases performance because you will use the rows; thus, access to them results in a cache hit.
- ◆ **Clusters.** Depending on the application, you may benefit from clusters. The benefit you receive depends on your data access patterns and your application.
- ◆ **Hash clusters.** Because batch processing typically does not use equality queries, a hash cluster will most likely degrade performance. Hash clusters are useful when your queries involve equality statements on the hash key.
- ◆ **Indexes.** Usually, batch processing systems depend heavily on the use of indexes. How and when you use indexes depends on your application. If you *do* use indexes, you may want to take advantage of the **FULL** hint to bypass the index when you know you will be doing a table scan.
- ◆ **Direct write sorts.** Direct write sorts is a new feature in Oracle that allows you to bypass the SGA when performing sort operations. Doing so prevents sorts from taking unnecessary space in the SGA; the result is more SGA space for other operations.
- ◆ **Parallel Query option.** The Oracle Parallel Query option will almost certainly improve the performance of your batch processing system. Parallel queries can be performed on sorts, joins, and table scans and improve the performance of those operations.

Many of these enhancements can help performance in a batch system. I recommend the use of the Oracle Parallel Query option, if at all possible. The specific effect on your system will vary, but most of the hints in this chapter work well for most situations.

Parallel Query Option

A DSS or batch processing system is probably the best application to take advantage of the Parallel Query option. The design of the Parallel Query option is centered around large

queries that access large amounts of data, as is typical of both DSS and batch processing systems. Here is a short list of some of the operations that can be parallelized:

- ◆ Sorts
- ◆ Joins
- ◆ Table scans

Depending on your application, you may use one or more of these transaction types. Because these queries are typically long running and involve large amounts of disk I/O, you can drastically cut the time it takes to complete these queries with the use of the Parallel Query option.

Once you set up a test database and develop some test queries (hopefully similar to, if not exactly like production queries), run these queries with various degrees of parallelism to determine optimal performance.

Because batch jobs are often very repeatable, any tuning changes that are beneficial in your test environment are often just as beneficial in production.

The Parallel Query option can take advantage of the concurrent nature of disk arrays. By splitting up the queries into many small pieces, you may be able to request data from all the drives in your system simultaneously, optimizing I/O performance.

Oracle Parallel Server Option

The Oracle Parallel Server option may be very beneficial to your batch processing system in two ways:

1. **Performance.** If your system is a candidate for the Parallel Server option, you should see significant performance improvements.
2. **Fault tolerance.** By using the Oracle Parallel Server option, you can keep the system running, even if a computer fails. Keep in mind that batch processing systems do not typically demand the same uptime requirements as OLTP systems.

The Oracle Parallel Server option can enhance the performance of your batch processing system—but only if queries can be divided among the different systems in the cluster. In general, I do not think that the Oracle Parallel Server option is particularly useful in a batch processing system. Parallel server clusters are better suited for OLTP or decision support systems.

NOTE: The Oracle Parallel Server option is not available on all platforms. Consult your system provider or Oracle to determine which platforms support the Oracle Parallel Server option.

Hardware Enhancements

In a batch system, several hardware enhancements can help you improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements—assuming that you are not I/O bound and that you are taking advantage of the Oracle Parallel Query option. The speed of CPUs is constantly being improved as are new and better cache designs.

For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before you purchase an additional processor of the same type and speed, consider upgrading to a faster processor. Because some batch processing jobs consist of a single job processing the data, you may benefit by purchasing a faster processor rather than more processors. Of course, this depends on your batch jobs and use of the Parallel Query option (refer to Chapter 13, “System Processors”).

SMP and MPP computers provide scalable CPU performance enhancements at a fraction of the cost of another computer. When upgrading your processors or adding additional processors, remember that your I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when you add processors.

I/O Enhancements

You can enhance I/O by adding disk drives or purchasing a hardware disk array. Batch processing systems can benefit from the disk striping available in hardware or software disk arrays, but the benefit of the disk array is enhanced by the use of the Parallel Query option. Using Oracle data file striping also helps the performance of your batch processing system.

Hardware and software disk arrays have the added benefit of optional fault tolerance. As described in Chapter 15, “Disk Arrays,” each of the fault-tolerant RAID levels has its advantages and disadvantages. You should first choose the correct fault tolerance for your needs and then make sure that you have sufficient I/O capabilities to achieve the required performance level. If you use fault tolerance, you will most likely have to increase the number of disk drives in your system.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If your queries often perform table scans, you may see good improved performance with disk controllers that take advantage of read-ahead features.

Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the batch processing system, it is likely that you will need that data soon; if you do, it will be available very quickly.

CAUTION: It is important to make sure that any controller you use with a write cache is protected against a power failure. Some disk array controllers on the market today (such as the Compaq disk array controllers) offer a battery-backed, mirrored memory cache that protects your data. Remember: Once Oracle believes that data has been written to the drive, that data had better be there.

Enhancements to the I/O subsystem almost always help in a batch processing environment (as they do with all RDBMS environments). Be sure that you have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

Network Enhancements

Network performance is never an issue during the actual transaction processing phase of the batch job, but it can be an issue during the load and unload phases of the batch job. Because large amounts of data may be transferred across the wire during these times, a high-speed link is beneficial.

You may be able to take advantage of the new 100Base-T technology or you may benefit from a fiber-optics link. The faster the network link, the faster the load/unload happens. You may find it beneficial to install a private link between the batch processing system and the location from which you are loading your data. Try to use the latest technology—such as 100 megabit Ethernet—to provide for future expansion.

Miscellaneous Enhancements

You may be able to take advantage of other hardware or software enhancements to improve the performance of your batch processing system:

- ◆ **High-speed tape.** If you load from tape or unload to tape, you may be able to take advantage of new technology such as DLT tapes or new high-speed tape devices. By enhancing the load/unload phase, the time to run the entire batch job is decreased because loading and unloading is all part of the batch job.
- ◆ **High-speed compression.** If part of your load/unload process requires compressing, moving to the latest hardware/software compression scheme can cut down on the time it takes to perform this compression.

Other miscellaneous enhancements may be available. See what is available on the market and determine whether it can help you.

Summary of System Enhancements

By taking advantage of some of the performance enhancement features available, you may see significant performance improvement. I have worked with the Oracle Parallel Query option and am very impressed with the performance gains you can achieve. Only you can determine how well your transactions can be parallelized.

Using the Parallel Query option in conjunction with a disk array is quite beneficial to your system's performance. By combining these enhancements with other tuning options such as a large block size and multiblock reads, you are on your way to having a well-tuned batch processing system.

Performance Verification

To verify that the changes you have made have positively affected performance, you must develop some sort of test plan. This test plan should be made up of queries very similar to—if not exactly the same as—the transactions you use on a daily basis. The closer your tests are to the actual transactions, the more accurate the performance verification is.

A good documented test plan allows you not only to verify system performance but to try new things, such as the Parallel Query option, direct write sorts, or some other new feature you may want to implement in the future. The following sections look at what should be tested in the RDBMS and the operating system.

What To Test in the RDBMS

To verify the performance of a batch processing system, the RDBMS should be tested to verify the load, processing, and unload times. Although a batch processing system does not have to perform with a strict response time criteria, often the maximum run time of the job is specified. To verify that you can meet this criteria, you should use a database that is close in size to the production database.

Once you have built an effective load test with repeatable performance results, start changing some of the parameters described in Part II of this book and earlier in this chapter. By changing only one parameter at a time, it is easier to see which changes have affected the performance.

Before changing anything, set your expectations for the changes you are going to make. If the results don't turn out as expected, try to understand why they didn't. Be sure to log the results so that you have a reference of what changes were effective and what changes were not.

What To Test in the OS

For the most part, the OS is tested with the RDBMS. If you have to change some specific OS parameter to increase a limitation, you usually don't have to retest the performance. Any limit change is usually associated with an RDBMS change, and the two can be tested together.

Other OS changes such as feature enhancements should be tested with the RDBMS, but you should more closely watch and analyze these results. You should carefully analyze things such as scheduler changes and cache affinity that may have an adverse affect on the system to verify that performance has not degraded.

As stated earlier in this book, the OS is primarily a vehicle for the RDBMS to use. The primary goal in tuning the server OS is to reduce overhead and optimize the I/O, memory, and networking subsystems.

Benchmarks

Standardized benchmarks are a good way to judge how well a system is performing and to compare different hardware platforms. If they are available to you, standardized benchmarks are also a good way to analyze the performance of your particular platform.

The TPC-C benchmark is an OLTP benchmark; the TPC-D benchmark is a strictly decision support benchmark. Batch processing systems require parts of both of these benchmarks. By looking at TPC benchmarks submitted for the platforms in which you are interested, you may be able to make some comparisons. Base your comparisons on a combination of the TPC-C and TPC-D benchmarks.

Examine the Full Disclosure Report (FDR) submitted for a TPC-D benchmark by the test sponsor; look for a breakdown of performance based on query type. Look for the performances for the particular query types you use in your operation. This information may give you an indication of how well the testing configuration would work in your environment. The TPC-C benchmark can give you an idea of how general OLTP processing performs on that platform. By looking at both of these benchmarks, you can get an idea of the performance of both the OLTP and DSS systems. Because batch jobs have some of the characteristics of both of these systems, these results may be of some value.

A system tuned for benchmarks such as the TPC-C and TPC-D benchmarks may provide useful tuning hints. Because the primary goal of a TPC benchmark is for the sponsor to get the best possible performance, you can see how the sponsors have optimally tuned their systems.

Because official TPC-C and TPC-D benchmarks using Oracle are usually submitted by the hardware manufacturers, OS vendors, and Oracle, you can be assured that the system has been tuned as optimally as possible. If you can, obtain a Full Disclosure Report from the TPC (on the Web at this URL: <http://www.tpc.org>); look at the way the system was designed and tuned.

Of course, the best benchmark to use on your batch processing system is one of your own batch processing jobs. This benchmark provides you with all the information you need to judge the amount and size of the hardware required. Such a customized benchmark also can provide you with valuable information about any configuration changes you make.

Summary

This chapter looked at some of the attributes of a batch processing system, explained how to analyze the characteristics of such a system, and suggested how to design an optimal configuration based on those attributes.

A batch processing system is different from OLTP and decision support systems but has some of the characteristics of both of them. A batch system typically involves loading data, processing that data, and unloading the results of the transactions.

Batch processing systems are different from OLTP systems in that batch jobs are not run interactively; they are submitted at some later time (perhaps much later) and run until completion. The characterization of the batch processing system in this chapter is generalized because batch processing systems vary in so many different ways. By looking at the attributes of your own system, you can better determine which of the tuning tips in this chapter will work for you and which will not.

The types of queries used in your batch processing system may be more representative of OLTP queries, be more like DSS queries, or may be something completely different. What is important is that you look at the queries themselves and examine the data access patterns; from that analysis, determine how your system can best be designed.

I am a big fan of the Oracle Parallel Query option (as you may have ascertained from my comments in this and other chapters). I believe that you will find a significant performance boost with the Parallel Query option if you have long-running, complex queries—especially those that involve table scans—a well-designed I/O subsystem, and a disk array.

Frequently, a batch processing system is used to run the same job over and over again with new data. This can be advantageous because it allows you to try new tuning parameters and be innovative. By having essentially the same job to run, you can quantitatively measure the results of the changes you make and determine whether performance has changed (either positively or negatively). By logging the results of these tests, you can gather valuable data that can help determine which changes you should try in the future.

Chapter **8**

Decision Support System

The decision support field has been growing in the last few years. Information that marketing and sales personnel had only dreamed of a few years ago is now available at their fingertips. Decision support is becoming a mainstream field.

A decision support system (DSS) uses the wealth of data usually gathered with OLTP systems to help you better meet market needs. A DSS can be used to perform many of the following business tasks:

- ◆ **Pricing and promotions.** You can determine when to offer promotions and at what price by looking at customer buying habits.
- ◆ **Supply and demand management.** By looking at data of which products sell and when, you may better be able to forecast supply and demand.
- ◆ **Profit and revenue management.** Trend analysis of expenses and profits can help you make better business decisions.
- ◆ **Customer satisfaction study.** You can determine trends and results of these studies based on many different criteria.
- ◆ **Market share study.** You can analyze the market based on the entire market by region or by organization.
- ◆ **Shipping management.** A wealth of data is at your fingertips.

These are just a few examples of the kind of activities available to you with this new, enabling technology.

NOTE: When I refer to decision support as *new*, I am not trying to give the impression that DSS systems have not been around for a while. In fact, DSS systems have been in existence for quite some time. DSS is new in the sense that an upsurge of activity in the area of DSS has been happening over the last few years.

The increased use of DSS systems has been helped in part by the availability of new high-performance, low-cost systems that make DSS available to more of the market than ever before. DSS class machines are now available for well under \$1 million and the performance of these machines is continually increasing.

In the next few years, DSS machines will be in use in areas that have not even been thought of yet. I believe that this will happen at both the very high end of computer technology and at the mid-range level. At the high end, computer technology will provide for faster CPUs and larger systems than ever before. The improved hardware will allow DSS systems to be designed to solve business problems that cannot be done with today's technology. Problems that would have taken months to compute will be done in days or hours.

At the other end of the spectrum are small businesses that do not have access to large systems; these businesses will purchase mid-range servers at reasonable prices to perform DSS functions. These businesses will become much more effective with this valuable tool.

The OLTP and batch systems are used to keep your business running on a day-to-day basis; the DSS system is used to predict trends and enable you to make sound business decisions based on the most up-to-date and accurate information, delivered in a form you can use.

NOTE: I realize that the word *system* is already in the DSS acronym (decision support system), but to simplify the flow of the book, I sometimes refer to a decision support system as a *DSS system*. Forgive the redundancy.

Characteristics of a DSS System

This section looks at some of the characteristics of a typical DSS system—if there is such a thing as a typical system. The following list generalizes some of the properties of a DSS:

- ◆ **Queries against large volumes of data.** The DSS system usually stores much more data than the typical OLTP system.
- ◆ **Queries exhibit a variety of access patterns.** Queries may be simple or quite complex with complicated joins and aggregations on large amounts of data.
- ◆ **Queries are of an ad-hoc nature.** This occurs because of the changing nature of the business models.
- ◆ **Highly complex queries.** Queries in a DSS system are far more complex than those in an OLTP system.
- ◆ **Generates intense activity on the part of the server components.** All aspects of the computer system are stressed during DSS queries.
- ◆ **Usually implemented to stay synchronized with an online production database.** The OLTP database is typically where the data is derived. As that database changes, the DSS database must keep up to date with it.

These characteristics typically make the load on the DSS system very high. In fact, because users do not typically use a DSS system for online processing, it is reasonable to push the system to its limits. Rather than spreading out the load and trying to achieve reasonable response times (as you do in an OLTP system), the goal of the DSS system is to push the system to its limits to get the work done as quickly as possible.

Because of the amount of data being queried and the complexity of those queries, it is not uncommon for DSS queries to take hours or even days. Do not expect to sit at your monitor, waiting for the DSS queries to respond. **Note:** By maximizing throughput, some jobs may suffer in terms of response time.

Data Access Patterns

For individual transactions, the data access patterns in a DSS system are fairly straightforward. Based on the types of transactions you are generating, you should be able to fairly accurately determine these patterns. Although each system has its own specific data access patterns, a decision support system has the following general characteristics.

- ◆ **Data access to the redo logs is either minimal or none.** If the system is purely used for DSS, there are usually no updates while DSS queries are running.
- ◆ **No archiving.** Archiving is not necessary because of the lack of activity on the redo logs.
- ◆ **Data access for each query is mostly sequential.** Because the nature of the queries is usually to extract large amounts of data from the tables, full-table scans are not uncommon.
- ◆ **Data reads can (and frequently do) take advantage of multiblock reads.** You can expect that many of the disk accesses are the size of multiblock reads.
- ◆ **Data access to the data files is somewhat random.** This random access is primarily caused by contention with other users and because join and index operations result in fairly random access across the data volumes. However, these random reads from the disk drives access the data with a much larger disk request.
- ◆ **Heavy access to the temporary tables.** This heavy access pattern is caused by the typical size of many of the join and sort operations. Remember: Only sorts that use less memory than `SORT_AREA_SIZE` are in memory.
- ◆ **Fairly even access to all data.** Although some data may be accessed more than others, this is usually on a table-by-table basis. Most data within the tables is accessed somewhat uniformly.

These patterns vary depending on how your system operates, but the general principles are the same. The access patterns to your tables vary based on how often the table is accessed and how much is done to each table.

System Load

In the DSS system, the CPUs usually are 100 percent active during the DSS queries. Unlike OLTP systems, which have many users with small queries, a DSS system has relatively few users and massive queries. These queries should be able to take advantage of the full capabilities of the CPUs and memory (as long as the system does not become disk bound). By following the guidelines described in Chapters 9 and 10, you can make the system CPU bound.

In the real world, I see systems that show a significant I/O deficiency. If you have an I/O capacity that does not fit the system load, you will most likely see a significant number of CPU idle cycles while the processes are waiting for I/Os to complete. Waiting for I/Os to complete can severely degrade performance.

The load characteristics of a DSS system include the following items:

- ◆ **A relatively few number of processes on the system.** This is true unless you take advantage of the Parallel Query option, which adds more processes and subsequently more process switches.

- ◆ **Minimal network traffic.** Typically, there is very little network traffic during the time that the DSS queries are running.
- ◆ **Heavy I/O usage.** DSS systems usually generate large amounts of I/O to the data files. This I/O is somewhat random if multiple DSS queries are active simultaneously but the I/Os are larger in size because of multiblock reads.
- ◆ **Very little or no redo log usage.** Because DSS functions typically do not do updates, the redo log is idle.
- ◆ **Very little or no use of rollback segments.** Again, because DSS functions consist of very little update activity, the rollback segments are hardly used.
- ◆ **Moderate amounts of memory.** The memory is used not only for the SGA but for each of the server processes for sort and join operations.

You can use these characteristics to help design and tune your DSS system for optimal performance. The first step in this design process is to set goals for what you want to achieve.

Goals

The goal in tuning the DSS system is to achieve a system that has certain characteristics. Here are the goals of an optimally tuned DSS system:

- ◆ **The system is CPU bound.** If you remove all other bottlenecks, the system should be able to process as fast as it can, which means at the speed of its CPUs.
- ◆ **The system does not show signs of being drive bound.** Any disk bottlenecks degrade performance. If the system is disk bound, you should either add more disks or increase memory.
- ◆ **Memory is sufficient.** If the machine pages or swaps, performance is severely degraded. The best solution is to add more memory; if that is not possible, reduce the size of the SGA or the number of users until the system no longer pages or swaps.
- ◆ **The system must meet any additional requirements you have.** With some DSS machines, it is necessary to keep current with the OLTP systems on a nightly basis. If this is a criterion of your system, you must make sure that you can upload new data within the specified time.

By setting goals for how you expect the system to perform, you can determine whether your tuning efforts are successful. You can also determine earlier whether you will be able to achieve the specified goals.

Review of DSS System Characteristics

By analyzing the characteristics of the DSS system along with the goals you want to achieve, you can obtain a lot of data with which you can build the system. Much of the design of the system is determined by how data is accessed and how many and what type of queries are specified.

The data access in a DSS system is generally of a random nature, although it does have sequential components. By configuring the system to take advantage of multiblock reads, you can achieve a high level of I/O performance.

Relate the information in the next part of this chapter to your particular configuration. Look for similarities and differences between what is described here and what you have observed about your system and decide how you can benefit from the tuning guidelines presented here. Although each system is different, many of the concepts remain the same.

You should base your system design on what you know about your application and system. Spend the time up front to carefully examine how the system needs to run so that you can determine the design of the system. The quality of the end product depends heavily on the amount of effort you put in at the beginning stages. No amount of tuning after the system is in production can make up for poor design choices.

Design Considerations

By looking at the data access patterns for your system, you should have a good idea of how the system operates. Before looking at the design of the system, here is a review of a few concepts introduced in earlier chapters:

- ◆ **I/Os are typically the limiting factor in the system.** You can only do a fixed number of random I/Os per second per disk drive (refer to Chapter 14, “Advanced Disk I/O Concepts”).
- ◆ **I/Os can be reduced by caching data blocks in the SGA.** If the data you want to access is already in the SGA, a disk I/O is not required.
- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out among many drives, you increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

All these factors contribute to the optimal data layout of the system. The physical layout—along with SGA and shared pool tuning—creates an optimally configured server for decision support tasks. In decision support systems, the design of the queries is also very important, as you will see in Part IV of this book, “Tuning SQL.”

Physical Data Layout

This section looks at how the data on a DSS system should be configured. First, it looks at how to lay out the data on traditional disks; then it looks at disk arrays. I recommend using disk arrays if at all possible; the ease of use and performance benefits are worth the cost of the array.

The main goal in designing the physical data layout is to balance the I/O across all the disks that are randomly accessed and to isolate the sequential I/O. Because the DSS system does not log very much, there isn't much sequential I/O to worry about. Because there is an insignificant amount of updating, the redo log files and archive logs do not see much use. You also know that the majority (if not all) of the data files are accessed in a random fashion but can take advantage of multiblock reads.

Many of the concepts presented here represent the best-case scenario. Budgetary constraints may not allow everyone to buy the optimal number of disks for their DSS configuration. Make the best of these guidelines. Remember that it is the number of disks that provide performance with random I/Os; if you have the opportunity to buy one 4 gigabyte disk drive or two 2 gigabyte disk drives, the best performance comes from the two 2 gigabyte disk drives.

Traditional Disks

The layout for a typical DSS system is fairly straightforward. The minimal configuration should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries.
Redo log (0)	Because very little update activity is involved with decision support activities, the log files are not active and are not a bottleneck to the system. These log-file drives can be better used for data files. The redo log files can reside on the same disk as the system.
Redo log (2+)	If online updates are done to the system, you must separate the redo log files, both for protection and performance. Make your decision based on the extent of the updates you plan to do.
Archive logs (0)	As with the redo log files, archiving is minimal and not necessary for a DSS system. The archive log files can also reside on the system disk.
Archive logs (1+)	As with the redo log files, if updates are done on this system, you must provide for the sequential nature of the archive log files.

continues

<i>Element (# of Disks)</i>	<i>Comments</i>
Data files (1+)	The number of disks you need for data is determined by the amount of random I/O your user community generates.
Index files (1+)	The number of disks needed for indexes is determined by the size of the indexes and the number of I/Os to the indexes.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

As you have seen in previous chapters, the data and indexes can be striped across the disks using Oracle or RAID striping or a combination of the two. With OLTP and batch processing systems, I recommended OS or hardware striping (as is the case here). But unlike OLTP or batch processing systems, in order to accommodate the Oracle Parallel Query option, it is important that queries are more optimally divided if you have several large extents. If you do not use Oracle striping and build one large extent, you may not see the full benefits of the Parallel Query option. So I recommend using OS or hardware striping, but divide your tablespace into multiple data files, each with several large extents, to accommodate parallel query processing.

I prefer a hardware disk array to manual Oracle striping primarily because the disk array provides excellent performance and is easy to use. When you use a disk array, the task of distributing I/Os can be greatly simplified.

Disk Arrays

The layout for the DSS system on a disk array is much simpler than it is on traditional disk drives. The minimal configuration for a DSS system on a disk array should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance.
Redo log (0)	<i>Little or no update activity.</i> Because very little update activity is involved with decision support activities, the log files are not active and are not a bottleneck to the system. These drives can be better used for data files. The redo log files can reside on the same disk as the system.

<i>Element (# of Disks)</i>	<i>Comments</i>
Redo log (1+)	<i>Some update activity.</i> You should have at least one log volume. This volume should be made up of at least two physical disks using RAID-1. If you use only one log volume, performance is degraded during archiving because the sequential nature of the log writes is disrupted.
Archive logs (0)	<i>Little or no update activity.</i> As with the redo log files, archiving is minimal and not necessary. The archive log files can also reside on the system disk.
Archive logs (1+)	<i>Some update activity.</i> The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary.
Data and index (1+)	<i>Because you always get concurrent access to disks with a disk array, you do not have to split the data and indexes into separate volumes.</i> The number of disks you need for data and index is determined by the amount of random I/O your user community generates.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

Unlike traditional disk drives, when you use a disk array, the data is automatically striped across all the disk drives; therefore, it is necessary to create only one tablespace and table for all your data. You do not even have to put indexes into another tablespace—although I recommend doing so for other reasons (such as monitoring and maintenance).

With traditional disk partitioning, it is difficult to manage hundreds of data files and disks; with a disk array, you can manage hundreds of disks with just a few data files. Of course, Oracle has a 2-gigabyte limitation on the size of a data file, but this is easily resolved by creating a data file for every 2 gigabytes of space you need. The data files can all reside on the same disk array volume. By splitting tablespaces into several data files with tables striped across them, all residing on the same logical volume, you can take better advantage of the Parallel Query option.

If you use a disk array, many of the management tasks and load balancing tasks are greatly simplified. With the disk array, you also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

I recommend that you use a disk array if possible. Software striping is fine, but if your system is under heavy loads (as it is with a typical DSS system), you can achieve better performance by offloading the striping overhead to a hardware RAID controller.

Hardware Considerations

When choosing hardware to use with a DSS system, consider these factors:

- ◆ **Low user load.** Not many concurrent processes/threads simultaneously access the system unless you are taking advantage of the Parallel Query option.
- ◆ **High I/O load.** I/Os are concurrent and heavy, with mostly random I/O.
- ◆ **Low network traffic.** The typical DSS does not use the network in any significant fashion (unless it is used for data loading).

If you can take advantage of the Oracle Parallel Query option, many different processes will use the machine at once; an SMP or MPP machine should scale very well. Because an SMP architecture uses CPUs based on the processes that are available to be run, if you always have a runnable process available for each CPU, you should see good scaling by adding additional processors. With an MPP machine, you see a similar effect but on a much larger scale.

Because there is much random access to the disks, you can benefit from a disk array. I prefer hardware striping to OS striping because hardware striping does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is a good alternative.

Because network traffic is probably not very high, you really don't have to worry too much about segmenting the network. If you are constantly updating the DSS system from the OLTP system, however, network traffic may be more of an issue.

Tuning Considerations

The DSS system is tuned to allow several large processes to run at maximum throughput. There is little concern for response times. All the tuning tips presented in Part II of this book, "Tuning the Server," apply here. Remember that with the DSS system, much less concern is given to user-response times; throughput counts.

You may have to tune both Oracle and the server operating system. The following sections look first at Oracle and then at the server operating system.

Oracle Tuning

Carefully analyze these things to determine whether adjustment to your Oracle parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** In the DSS system, block size is not nearly as critical as it is in OLTP and batch systems. However, having an insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck. The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The ratio of PHYSICAL READS to DB BLOCK GETS and CONSISTENT GETS is the cache-miss ratio. This number should be minimized.

- ◆ **Library cache.** Remember to check The V\$LIBRARYCACHE table that contains statistics about how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You can reduce the library cache misses by increasing the amount of memory available for the library cache. Do this by increasing the Oracle tunable parameter SHARED_POOL_SIZE. Try increasing the size of the shared pool by 10 percent and monitor it again. If this is not sufficient, increase the size of the shared pool by another 10 percent until you are pleased with the performance.
- ◆ **Multiblock reads.** Because many DSS queries involve table scans, make sure that you can take advantage of multiblock reads. The number of blocks read in a multiblock read is specified by DB_FILE_MULTIBLOCK_READ_COUNT, an Oracle initialization parameter. This value, multiplied by the DB_BLOCK_SIZE parameter, results in the size of the I/Os. A good value for the size of a multiblock read I/O is 64K.
- ◆ **Cursor space for time.** If you have plenty of memory, you can speed access to the shared SQL areas by setting the CURSOR_SPACE_FOR_TIME Oracle initialization parameter to TRUE.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, look at the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback segments.** Because of the limited number of updates in a DSS system, rollback contention and the number of rollback segments are usually not important.
- ◆ **Latch contention.** Because of the limited number of updates in a DSS system, latches are also not an issue.
- ◆ **Checkpoints.** The effect of the checkpoint is not an issue in a typical DSS system because user response times are not critical.
- ◆ **Archiving.** As with the other log-related parameters, the limited updates in a DSS system mean that archiving is not an issue.

Pay particular attention to these areas when tuning a system for decision support. Probably the area that requires the most attention is I/O and memory because these two are so closely related. By optimizing the use of memory, you may be able to reduce I/Os (which are probably running near the limitations of the hardware).

Server OS Tuning

You may have to tune the server OS to provide optimal I/O performance. Some of the things you may have to tune in the server OS are listed here; remember that some OSes may not require any tuning in these areas:

- ◆ **Memory.** Tune the system to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes. You may also need significant amounts of memory for sorts.
- ◆ **Memory enhancements.** Take advantage of 4M pages and ISM, if they are available. Both features can improve Oracle performance in a DSS environment.
- ◆ **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- ◆ **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In a DSS system, allowing a process to run to completion (that is, so that it is not preempted) is beneficial.
- ◆ **Cache affinity.** You may see some benefits from cache affinity in a DSS system because the processes tend to run somewhat longer.

The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, you can enhance Oracle performance.

Enhancements

You may want to make some additional enhancements to improve the performance of the DSS system. Listed here are some of the items that can offer performance improvement; also listed are a few items that do not improve performance. Each item is accompanied by an explanation.

- ◆ **Block size.** In a DSS system, you may find that increasing the database block size can greatly improve performance. Because many of the queries access the database in a sequential manner, even though concurrent queries randomize the I/O, having a larger block size brings more of the rows into the SGA at once. Having these additional rows in the SGA can benefit you because you *will* be using them.
- ◆ **Clusters.** Depending on the application, you may benefit from clusters. The benefit you receive depends completely on the data access patterns and your application.
- ◆ **Hash clusters.** Because the DSS system typically uses table scans to get large amounts of data, a hash cluster will most likely degrade performance. Hash clusters are useful when the queries involve equality statements on the hash key.
- ◆ **Indexes.** Again, because table scans are involved in most queries, indexes may not be particularly useful (although some queries may benefit from the use of indexes). How and when you use indexes depends on your application. If you do use indexes, you may want to take advantage of the **FULL** hint, which bypasses the index, if you know you will be doing a table scan.
- ◆ **Direct write sorts.** Direct write sorts is a new feature that allows you to bypass the SGA when performing sort operations. This feature prevents sorts from taking unnecessary space in the SGA; the result is more SGA space for other operations.

- ◆ **Parallel Query option.** The Oracle Parallel Query option can almost certainly improve the performance of your DSS system. Parallel queries can be performed on sorts, joins, and table scans to improve the performance of those operations.

Many of these enhancements can help your performance; some may degrade performance. The specific effect on your system will vary, but the information given here is true for most general cases.

Parallel Query Option

The DSS system is probably the best application to take advantage of the Parallel Query option. The design of the Parallel Query option is centered around complex queries that access large amounts of data—as is typical of a DSS system. Here is a short list of some of the operations that can be parallelized:

- ◆ Sorts
- ◆ Joins
- ◆ Table scans

All these query types are typically used in a DSS system. Because these queries are typically of long duration and involve large amounts of disk I/O, the time it takes to complete these queries can be cut drastically with the use of the Parallel Query option.

Once you set up a test database and develop some test queries (hopefully similar to, if not exactly the same as, production queries), run these queries with various degrees of parallelism to determine optimal performance.

If you will run these queries with other queries in production, you should also test with multiple queries. A tuning parameter set up for use with a single query may not be optimal when used with many queries in parallel.

The Parallel Query option can take advantage of the concurrent nature of disk arrays. By splitting up the queries into many small pieces, you may be able to request data from all the drives in your system simultaneously, optimizing I/O performance.

Oracle Parallel Server Option

The Oracle Parallel Server option may be very beneficial to your DSS operation in two ways:

1. **Performance.** If your system is a candidate for the Parallel Server option, you will see significant performance improvements.
2. **Fault tolerance.** By using the Oracle Parallel Server option, you can keep the system running even if a computer fails. DSS systems do not typically have the same uptime requirements as OLTP systems.

The Oracle Parallel Server option can enhance the performance of your DSS system, but only if your system is suited for the Parallel Server option. To see scaleable performance increases, the system must meet the following criteria:

- ◆ **Partitionable data.** The data is partitionable. In other words, the work related to certain tables can be done on one computer while work related to other tables is done on another computer. If all users access the exact same data, your application may not be a good candidate for the Parallel Server option. Although some overlap is fine, the majority of the data should be partitionable.
- ◆ **Many processes.** If you have only a few users and concurrent transactions, you probably are not a good candidate for the Parallel Server option. However, if you use the Parallel Query option, you may use a sufficient number of query server processes to satisfy this requirement.

If these conditions are met, you will probably benefit from using the Oracle Parallel Server option.

NOTE: The Oracle Parallel Server option is not available on all platforms. Consult your system provider or Oracle to determine which platforms support the Oracle Parallel Server option.

Hardware Enhancements

For a DSS system, there are several hardware enhancements that can improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements, assuming that you are not I/O bound. The speed of CPUs is constantly being improved as are new and better cache designs.

For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before you purchase an additional processor of the same type and speed, however, consider upgrading to a faster processor. For example, upgrading from a 66 MHz processor to a 133 MHz processor may provide more benefit than purchasing an additional 66 MHz CPU—with the added benefit that you now have the option of adding more 133 MHz CPUs. This determination is usually made by budgetary constraints. Because the DSS can take advantage of the Parallel Query option, adding processors will help performance as well adding a faster CPU.

SMP and MPP computers provide scaleable CPU performance enhancements at a fraction of the cost of another computer. When upgrading your processors or adding additional processors, remember that your I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when you add processors.

I/O Enhancements

You can enhance I/O by adding disk drives or purchasing a hardware disk array. DSS systems can benefit from the disk striping available in both hardware and software disk arrays. Using Oracle data file striping can also help the performance of the DSS system.

If your system performs only one query at a time and you are not taking advantage of the Oracle Parallel Query option, you may not see a benefit from a hardware or software disk array. In this specific case, I do not recommend OS or hardware striping; you should use traditional Oracle striping. Because you are executing only one query at a time without using the Parallel Query option, the I/Os to the data files are purely sequential on the table scans. This scenario is somewhat rare; any variance from “pure table scans” results in degraded performance.

Hardware and software disk arrays have the added benefit of optional fault tolerance. As described in Chapter 15, “Disk Arrays,” each of the fault-tolerant RAID levels has its advantages and disadvantages. You should first choose the correct fault tolerance for your needs and then make sure that you have sufficient I/O capabilities to achieve the required performance level. If you use fault tolerance, you will most likely have to increase the number of disk drives in your system.

NOTE: Because DSS systems primarily involve a majority of read operations and very few writes, you can take advantage of the fault tolerance and read performance of RAID-5. Make sure that write activity is small and remember that there may be significant write activity in the temporary tables.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If your queries often perform table scans, you may see improved performance with disk controllers that take advantage of read-ahead features.

Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the DSS system, it is likely that you will need that data soon; if you do, it will be available very quickly.

CAUTION: It is important to make sure that any controller you use with a write cache is protected against a power failure. Some disk array controllers on the market today (such as the Compaq disk array controllers) offer a battery-backed, mirrored memory cache that protects your data. Remember: Once Oracle believes that data has been written to the drive, that data had better be there.

Enhancements to the I/O subsystem almost always help in a DSS environment because large amounts of data are read. Be sure that you have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

Network Enhancements

Networks are usually not a problem in a DSS system. However, because the data used in your DSS may be extracted from the OLTP systems in your company, you may have some network issues. By synchronizing the OLTP and DSS systems during off hours and using large block transfers, you may be able to avoid network traffic issues.

If necessary, you may want to install a private link between the DSS system and the OLTP systems that provide data. If you decide to do this, try to get the latest technology—such as 100 megabit Ethernet—to provide for future expansion.

Review of Enhancement Options

By taking advantage of some of the performance enhancement features available, you may see significant performance improvement. I have worked with the Oracle Parallel Query option and I am very impressed with the performance gains possible. DSS is particularly well suited for the parallelizing of queries.

Using the Parallel Query option in conjunction with a disk array can be beneficial to your system's performance. By combining these enhancements with other tuning options (such as a large block size and multiblock reads) you are on your way to having a well-tuned DSS system.

Performance Verification

Once you design and build your system, you must have some way of verifying its performance. Verifying a DSS environment is much easier than verifying an OLTP system because you do not have to simulate large numbers of users.

The simplest method is to test the system using production queries on a test database. By saving the test database and using the same queries on the same data, you can make direct comparisons when you change tuning parameters or upgrade the system.

A good documented test plan allows you not only to verify system performance but to try new things, such as the Parallel Query option, direct-write sorts, or some other new feature you may want to implement in the future. The following sections look at what should be tested in the RDBMS and the operating system.

What To Test in the RDBMS

You should test the RDBMS keeping functionality, load testing, and performance in mind. Make sure that the system will function as specified. Try to use a test database that is configured and sized similar to the production database.

Once you build an effective load test with repeatable results, start changing some of the parameters described in Part II of this book and earlier in this chapter. By changing only one parameter at a time, you can easily see which changes affect performance.

Before changing anything, set your expectations for the changes you are going to make. If the results don't turn out as expected, try to understand why they didn't. Be sure to log the results so that you have a reference of what changes were effective and what changes were not.

What To Test in the OS

For the most part, the OS is tested with the RDBMS. If you have to change some specific OS parameter to increase a limitation, you usually don't have to retest the performance. Any limit change is usually associated with an RDBMS change, and the two can be tested together.

Other OS changes such as feature enhancements should be tested with the RDBMS, but you should more closely watch and analyze these results. You should carefully analyze things such as scheduler changes and cache affinity that may have an adverse affect on the system to verify that performance has not degraded.

As stated earlier in this book, the OS is primarily a vehicle for the RDBMS to use. The primary goal in tuning the server OS is to reduce overhead and optimize the I/O, memory, and networking subsystems.

Benchmarks

Standardized benchmarks are a good way to judge how well a system is performing and to use as a comparison between different hardware platforms. If they are available to you, standardized benchmarks are also a good way to analyze the performance of your particular platform.

Using a standardized benchmark (such as the TPC-D benchmark) is not a good way to tune your system. You should tune each system individually, based on its own characteristics. Examine the Full Disclosure Report (FDR) submitted by the test sponsor to see a breakdown of performance by query type. Look at the performance for the particular query types you use in your operation. Doing so may give you some indication of how well the testing configuration would work in your environment.

A system tuned for a benchmark such as the TPC-D benchmark may provide useful tuning hints. Because the primary goal of a TPC benchmark is for the sponsor to get the best possible performance, you can see how the sponsors have optimally tuned their systems.

Because an official TPC-D benchmark using Oracle is usually submitted by the hardware manufacturers, OS vendors, and Oracle, you can be assured that the system has been tuned as optimally as possible. If you can, obtain a Full Disclosure Report from the TPC (on the Web at this URL: <http://www.tpc.org>); look at the way the system was designed and tuned.

Of course, the best benchmark to use on your DSS system is one of your own production decision-support queries. This benchmark provides you with all the information you need to judge the amount and size of the hardware required. Such a customized benchmark also can provide you with valuable information about any configuration changes you make.

Summary

The decision support field has been growing in the last few years. With the introduction of faster and less-expensive computer systems, more and more businesses can take advantage of decision support systems. In fact, decision support is now becoming very mainstream.

This chapter looked at the characteristics of this system from the perspectives of a business model and data access. In the next few years, DSS machines will be in use in areas that have not even been thought of yet. As the cost of computing continues to fall and the speed of computing increases, innovative ways of using DSS systems will emerge.

Chapter

9

Data Warehousing System

The concept of the data warehouse has been emerging over the last few years, but with the new, faster computers and the reduction in the cost of disk storage, the data warehouse is finally becoming a reality. This chapter first looks at what a data warehouse is, then at the characteristics of the data warehouse, and then at the data access patterns seen in the data warehouse.

As its name implies, the data warehouse is a storage depot for corporate data. Enormous amounts of data are concentrated in the data warehouse; sources for the data include the following:

- ◆ Customer information databases
- ◆ Accounts receivable
- ◆ General ledger
- ◆ Inventory databases
- ◆ Customer credit databases
- ◆ Other sources

These sources combine to provide a wealth of data about customers and their buying habits as well as information about the general state of your business.

A data warehousing system is similar to a DSS system in some of its functionality, but the scale and focus are different. A typical DSS system focuses on one type of business function; by using its various data-input sources, the data warehousing system may perform much broader business queries.

Data warehousing systems can easily achieve sizes in the hundreds of gigabytes; some systems even break the terabyte barrier. These systems are made possible by the continuing trend of computer hardware to increase in speed while decreasing in price. In the near future, it may not be uncommon to see tables of a terabyte in size.

Although the cost of data warehousing hardware is decreasing, it is still out of reach of the mainstream—for now. As we go further into the information age and the value of information is better understood, I believe data warehousing will become more mainstream.

Characteristics of a Data Warehouse

Here are some of the characteristics of the data warehousing system:

- ◆ **Queries against large volumes of data.** The data warehousing system consists of much more data than the typical OLTP or DSS system.
- ◆ **Queries exhibit a variety of access patterns.** Queries may be simple or quite complex, with complicated joins and aggregations on large amounts of data.
- ◆ **Highly complex queries.** Queries on a data warehouse are typically much more complex than those used in OLTP and DSS systems.
- ◆ **Data access stresses the system to its limitations.** The processes stretch the system in terms of both performance and capacity.
- ◆ **Intense load activity.** As data from various sources is entered into the data warehousing system, the load increases dramatically.
- ◆ **Is a conglomeration.** A data warehouse is a compilation of various input sources, usually tied into the corporate OLTP databases and other sources.

The load on the data warehousing system is typically very high. As with the DSS system, because users do not typically use a data warehousing system for online processing, it is reasonable to push the system to its limits.

It is not uncommon for the decision support queries run against the data warehousing system to take hours or even days to complete. The queries are complex and the amount of data being queried is enormous. These systems are optimized for throughput rather than for response times. By maximizing throughput, some jobs may suffer in terms of response time.

If you think that the data warehousing system is just a glorified DSS system, you are partially correct. The data warehouse may just be the next step in the evolution of the DSS system. There are many similarities, but there are many differences as well.

Data Access Patterns

The data access patterns seen in a data warehouse are fairly similar to those seen in a DSS system. Based on the types of transactions you generate, you should be able to fairly accurately determine these patterns. Although each system has its own specific data access patterns, the data warehousing system has the following, general characteristics:

- ◆ **Redo log activity is moderate to high.** Unlike the DSS system (where the redo log activity is very low), the redo log activity for a data warehouse may be moderate or even high. This is caused, not by the activity of the business transactions, but by the procedures necessary to prepare and load the data. The metadata may be constantly put together from many external sources.
- ◆ **Archiving activity is moderate to high.** As with the redo logs, there may be significant activity due to the conglomeration of data stored in the data warehouse.
- ◆ **Data access for each query is mostly sequential.** Because the queries usually extract large amounts of data from the tables, full-table scans are not uncommon.
- ◆ **Data reads can (and frequently do) take advantage of multiblock reads.** You can expect that many of the disk accesses are the size of multiblock reads.
- ◆ **Access to the data files is somewhat random.** As with the DSS system, this random access is caused by contention with other transactions and join and indexing operations that result in fairly random access across the data volumes. However, these random reads from the disk drives access the data with a much larger data request.
- ◆ **Data access may be sparse.** Because of the massive amounts of data being stored, there may be pockets of data infrequently accessed and other pockets that see much more frequent access.
- ◆ **Heavy access to the temporary tables.** Because of the typical size of many of the join and sort operations, the temporary tables are hit hard. Remember that only the sorts that use less memory than `SORT_AREA_SIZE` are in memory.

- ◆ **Data access may not be distributed evenly.** Because of the massive amount of data stored in the warehouse, it is not uncommon for queries to use only isolated pieces of data.

Although these patterns vary depending on how your system operates, the general principles are the same. The access patterns to your tables vary based on how often and how much is done to each table.

System Load

Like the DSS system, the CPUs in a data warehousing system are usually 100 percent active during the large business queries. Where OLTP systems have many users with small queries, the data warehousing system has relatively few users and massive queries (as does a DSS system). These queries should be able to take advantage of the capabilities of the CPUs and memory as long as the system does not become disk bound. By tuning the server using some of the concepts discussed in Chapters 9 and 10, you can avoid becoming disk bound.

The typical OLTP, batch, or DSS system may have an insufficient number of disk drives, which causes an I/O bottleneck. The data warehousing system may not have this problem because so much disk space is needed for the hundreds of gigabytes of historical and current business data, that, with careful planning, there are plenty of disk spindles to distribute the I/O load.

Following is a list of some of the load characteristics of a data warehousing system:

- ◆ **Relatively few processes on the system.** If you take advantage of the Parallel Query option, you add more processes and subsequently more process switches.
- ◆ **Minimal network traffic.** Network traffic is low during the transaction processing phases but may be significant during the data loading and updating phases.
- ◆ **Heavy I/O usage.** The decision support queries associated with the data warehouse usually generate large amounts of I/O to the data files. This I/O is somewhat random if multiple decision support queries are active simultaneously; but the I/Os are larger in size because of multiblock reads.
- ◆ **Moderate to high redo log activity.** Unlike the DSS system (where the redo log activity is very low), the redo log activity for a data warehouse may be moderate or even high. This is caused, not by the activity of the business transactions, but by the procedures necessary to prepare and load the data. The metadata may be constantly put together from many external sources.
- ◆ **Moderate to heavy use of rollback segments.** During the decision support queries, rollback segments will not be used heavily, but during the data-creation or conversion phase, rollback segment activity can be significant.
- ◆ **Large amounts of memory.** The memory is used not only for the SGA but for each of the server processes required for sort and join operations.

Defining and understanding these characteristics can help you design and tune your data warehouse for optimal performance. The first step in this design process is to set goals for what you want to achieve.

Goals

The goal in tuning the data warehouse is to achieve a system that has certain characteristics. Here is a list of the characteristics of an optimally tuned data warehouse:

- ◆ **The system is CPU bound during decision support queries.** By removing all other bottlenecks, the system should be able to process as fast as possible, which is the speed of the CPUs.
- ◆ **The system is not drive bound.** Any disk bottleneck degrades performance. If this is the case, you should add more or faster disks.
- ◆ **Memory is sufficient.** If the machine pages or swaps, performance is severely degraded. The best solution is to add more memory; if that is not possible, reduce the size of the SGA or the number of users until the system no longer pages or swaps.
- ◆ **The system meets any additional requirements you might have.** With some data warehousing machines, you must keep current with the OLTP systems by updating on a nightly basis. With the data warehouse, you may have to update the data within a certain time frame.

By setting goals for how you expect the system to perform, you can determine whether you are successful. You can also determine earlier whether you will be able to achieve your specified goals.

Review of Data Warehouse Characteristics

By analyzing how the system operates, how the queries work, and how the data is accessed, you are well on your way to determining the best configuration for your data warehouse. This information, together with the tuning procedures described in Part II of this book, can help you design an optimal configuration.

Data access in a data warehousing system is generally random, with some sequential components. By configuring the system and taking advantage of multiblock reads, you can achieve a high level of I/O performance.

Relate the information in the next part of this chapter to your particular configuration. Look for similarities and differences between what is described here and what you have observed about your system and decide how you can benefit from the tuning guidelines presented here. Although each system is different, many of the concepts remain the same.

You should base your system design on what you know about how your system operates and accesses data. Spend the time up front to carefully examine how the system needs to run so that you can determine the design of the system. The quality of the end product depends heavily on the amount of effort you put in at the beginning stages. No amount of tuning after the system is in production can make up for poor design choices.

Design Considerations

Looking at data access patterns can give you a good idea how to design the system. Before looking at the design process, consider these important issues, introduced earlier in this book:

- ◆ **I/O is typically the limiting factor in the system.** You can do only a fixed number of random I/Os per second per disk drive (refer to Chapter 14, “Advanced Disk I/O Concepts”).
- ◆ **I/Os can be reduced by caching data blocks in the SGA.** If the data you want to access is already in the SGA, a disk I/O is not required.
- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out among many drives, you increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

All these factors contribute to the optimal data layout of the system. The physical layout—along with SGA and shared pool tuning—creates an optimally configured server for the decision support tasks usually performed in the data warehouse. In data warehousing systems, the design of the queries is also very important, as you will see in later chapters.

Physical Data Layout

This section looks at how the data in a data warehouse should be configured. First, it looks at how to lay out the data on traditional disks; then it looks at disk arrays. I recommend using disk arrays if at all possible; the ease of use and performance benefits are worth the cost of the array.

The main goals in designing the physical data layout are to balance the I/O across all the disks that are randomly accessed and to isolate the sequential I/O. The data warehousing system typically involves loading and processing of data, which causes moderate to significant use of the redo logs. By isolating the redo log files to their own disk volumes, you can take advantage of the sequential nature of their I/Os.

In a data warehouse, the majority (if not all) of the data files are accessed in a random fashion but can take advantage of multiblock reads. To take advantage of multiblock reads, stripe the data over as many disks as necessary to achieve I/O rates your disk drives can handle.

In other chapters of this book, I have recommended that you use smaller disks rather than larger ones to maximize disk count. I do not recommend that here. Because of the enormous amounts of data being stored in the data warehouse, it is probably most economical to purchase fairly large disks. Don't go overboard and buy only a few enormous disks, but don't purchase the smallest ones either.

Traditional Disks

The layout for a data warehouse can be large and difficult to manage. A minimal configuration should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries.
Redo log (2+)	Because there is moderate to heavy redo log activity, it is best to have at least two disk drives so that you can mirror the logs. When you factor in archiving, you may be better off with at least four disks.
Archive logs (1+)	You can take advantage of the sequential nature of the archive process by isolating the archive log files to their own set of disks.
Data files	The number of disks you need for data is determined by the amount of random I/O your user community generates and the size of the database. In this type of environment, the number of disks can be significant.
Index files	The number of disks needed for indexes is determined by the size of the indexes and the number of I/Os to the indexes. In this type of environment, the number of index disk drives can also be significant.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

As you have seen in previous chapters, the data and indexes can be striped across the disks using Oracle or RAID striping or a combination of the two. With large data warehousing systems, I recommended OS or hardware striping. To take advantage of the Oracle Parallel Query

option, you will benefit from having several large extents. An optimal configuration may consist of several data files residing on the same large, striped volume. If you do not use Oracle striping and build one large extent, you may not see the full benefits of the Parallel Query option.

I prefer a hardware disk array to manual Oracle striping primarily because the disk array provides excellent performance and is easy to use. When you use a disk array, the task of distributing I/Os can be greatly simplified.

Disk Arrays

The layout for the data warehouse on RAID volumes is much simpler than it is on traditional disk drives. A minimal configuration should look something like this:

<i>Element (# of Volumes)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, you should mirror this disk for additional fault tolerance.
Redo log (2+)	Because there is moderate to heavy redo log activity, it is best to have at least two disk drives so that you can mirror the logs. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance degrades during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you need to archive. This data can be written to tape as necessary. You can take advantage of the sequential nature of the archive process by isolating the archive log files to their own set of disks.
Data and index (1+)	Because you always have concurrent access to disks in a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates and the size of the database. This logical volume may consist of many disk drives. Performance is enhanced by using as few volumes as possible and striping over as many drives as possible.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

Unlike traditional disk drives, when you use a disk array, the data is automatically striped across all the disk drives; therefore, it is necessary to create only one tablespace and table for all your data. You do not even have to put indexes into another tablespace—although I recommend doing so for other reasons (such as monitoring and maintenance).

With traditional disk partitioning, it is difficult to manage hundreds of data files and disks; with a disk array, you can manage hundreds of disks with just a few data files. Of course, Oracle has a 2 gigabyte limitation on the size of a data file, but this is easily resolved by creating a data file for every 2 gigabytes of space you need. The data files can all reside on the same disk array volume. By splitting tablespaces into several data files with tables striped across them, all residing on the same logical volume, you can take better advantage of the Parallel Query option.

Because the data warehouse may have a large amount of data that is sparsely accessed, it is to your advantage to put many different types of archival and current data on each disk volume. By spreading out the data, the I/O load is more evenly distributed.

If you use a disk array, many of the management tasks and load balancing tasks are greatly simplified. With the disk array, you also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

I recommend that you use a disk array if possible. Software striping is fine, but if your system is under heavy loads (as it is with a typical data warehousing system), you can achieve better performance by offloading the striping overhead to a hardware RAID controller.

Fault Tolerance Consideration

Because the data warehouse contains so much data, you can take one of two approaches to data protection:

- ◆ **Protect everything.** Because there is so much data and so many disks in use, everything must be protected. The large number of disks in use increases the possibility of a disk failure. The massive amount of data increases the time needed for backup and recovery.
- ◆ **Conserve cost.** Because there are so many disks involved, it may be cost prohibitive to use RAID-1 or disk mirroring. When you mirror the disks, you double the number of disks.

In a data warehousing system, a good compromise is to use a fault tolerant method such as RAID-5 for the data files. You can be somewhat selective and use RAID-1 on volumes with heavy update activity and RAID-5 on volumes with more read activity. Remember that the performance penalty for RAID-5 is only on writing; you can achieve excellent read performance from RAID-5.

Hardware Considerations

When choosing hardware to use for a data warehousing system, consider these factors:

- ◆ **Low user load.** Not many concurrent processes/threads simultaneously access the system—unless you take advantage of the Parallel Query option.
- ◆ **High I/O load.** I/Os are concurrent and heavy, with mostly random I/O.
- ◆ **Huge amounts of data.** Data warehousing systems typically involve massive amounts of data. You must make sure that your system can support the high volumes of data you will be using.
- ◆ **Low network traffic during runtime, possibly high during load.** During the execution of typical decision support queries against your data warehouse, there is very little network activity. When data is being loaded or updated from other sources (possibly your OLTP systems), the network activity can be quite high.

If you can take advantage of the Oracle Parallel Query option, many different processes will use the machine at once; an SMP or MPP machine should scale very well. Because an SMP architecture uses CPUs based on the processes that are available to be run, if you always have a runnable process available for each CPU, you should see good scaling by adding additional processors. With an MPP machine, you see a similar effect but on a much larger scale.

Because there is much random access to the disks, you can benefit from a disk array. I prefer hardware striping to OS striping because hardware striping does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is adequate.

Network traffic may or may not be an issue to your data warehousing system. If necessary, segment the network or add faster network hardware. A network bottleneck is an easy problem to solve: simply add more and faster hardware.

Tuning Considerations

The data warehouse is tuned to allow several large processes to run at maximum throughput. There is usually no concern for response times. All the tuning tips described in Part II of this book, “Tuning the Server,” apply here. Remember that with the data warehouse, much less concern is given to user response times; throughput counts.

You may have to tune both Oracle and the server operating system. The following sections look first at Oracle and then at the server operating system.

Oracle Tuning

Carefully analyze these areas to determine whether adjustment to these parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** Block size is not nearly as critical in a data warehouse as it is in the OLTP and batch systems. However, having an insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck. The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The ratio of PHYSICAL READS to DB BLOCK GETS and CONSISTENT GETS is the cache-miss ratio. This number should be minimized.
- ◆ **Library cache.** Check The V\$LIBRARYCACHE table, which contains statistics about how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You should be able to reduce the library cache misses by increasing the amount of memory available for the library cache. Do so by increasing the Oracle tunable parameter SHARED_POOL_SIZE. Try increasing the size of the shared pool by 10 percent and monitor it again. If this is not sufficient, increase the size of the shared pool by another 10 percent and continue in this manner until you are satisfied with the result.
- ◆ **Multiblock reads.** Because many of the queries will involve table scans, make sure that you take advantage of multiblock reads. The number of blocks read in a multiblock read is specified by the Oracle initialization parameter DB_FILE_MULTIBLOCK_READ_COUNT. This value, multiplied by the DB_BLOCK_SIZE parameter, results in the size of the I/Os. A good value for the I/O size is 64K.
- ◆ **Cursor space for time.** If you have plenty of memory, you can speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR_SPACE_FOR_TIME to TRUE.
- ◆ **Data dictionary cache.** To determine the efficiency of the data dictionary cache, check the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback segments.** Depending on the number of updates, you may want to increase the number of rollback segments to reduce contention. Unlike an OLTP system, in data warehousing systems, you are better off with more rollback segments of a larger size. The size of the rollback segments depends on the size of the transactions. Because of the type of transactions usually associated with a data warehouse, your rollback segments should be fairly large. Rollback contention occurs when too many transactions try to use the same rollback segment at the same time; some of them have to wait. You can tell if you have contention on rollback segments by looking at the dynamic performance table V\$WAITSTAT. Check for an excessive number of UNDO HEADERs, UNDO BLOCKs, SYSTEM UNDO HEADERs, and SYSTEM UNDO BLOCKs. Compare these values to the total number of requests for data. If the number is high, you need more rollback segments.

- ◆ **Latch contention.** You can determine whether latch contention is a problem by examining the dynamic performance table V\$LATCH. Look for the ratio of MISSES to GETS, the number of SLEEPS, and the ratio of IMMEDIATE_MISSES to IMMEDIATE_GETS. If the miss ratio is high, reduce the size of LOG_SMALL_ENTRY_MAX_SIZE to minimize the time any user process holds the latch; alternatively, increase the value of LOG_SIMULTANEOUS_COPIES to reduce contention by adding more redo copy latches. If neither of these approaches helps, set the initialization parameter LOG_ENTRY_PREBUILD_THRESHOLD. Any redo entry of a smaller size than this parameter must be prebuilt, which reduces the time the latch is held.
- ◆ **Checkpoints.** You may have to tune checkpoints under certain circumstances. Although this is usually not necessary, if you see severely degraded performance during checkpoints, you can reduce this effect by enabling the CKPT process. Do so by setting the Oracle initialization parameter CHECKPOINT_PROCESS to TRUE.
- ◆ **Archiving.** By adjusting the initialization parameters LOG_ARCHIVE_BUFFERS and LOG_ARCHIVE_BUFFER_SIZE, you can either slow down or speed up the performance of archiving. By speeding up archiving, the effect on the system is of a shorter duration but is more noticeable. Slowing down archiving lengthens the duration but reduces the effect.

These are some of the areas you should pay particular attention to when tuning a system for decision support queries in a data warehousing system. Probably the areas that require the most attention are I/O and memory because these two are so closely related. By optimizing the use of memory, you may be able to reduce I/Os (which are probably running near the limitations of the hardware).

Server OS Tuning

You may have to tune the server OS to provide for a large number of processes (if you are using the Parallel Query option) and optimal I/O performance. Some of the things you may have to tune in the server OS are listed here; remember that some OSes may not require any tuning in these areas:

- ◆ **Memory.** Tune the system to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes. You may also need significant amounts of memory for sorts.
- ◆ **Memory enhancements.** Take advantage of 4M pages and ISM, if they are available. Both features can improve Oracle performance in a data warehouse environment.
- ◆ **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- ◆ **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In a data warehousing system, allowing a process to run to completion (that is, so that it is not preempted) is beneficial.

- ◆ **Cache affinity.** You may see some benefits from cache affinity in a data warehousing system because the processes tend to run somewhat longer.

The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, you can enhance Oracle performance.

Enhancements

You may want to make some additional enhancements to improve the performance of your data warehouse. Listed here are some of the items that can offer performance improvement; also listed are a few items that do not improve performance. Each item is accompanied by an explanation.

- ◆ **Block size.** In a data warehousing system, you may find that increasing the database block size can greatly improve performance. Because many of the queries access the database in a sequential manner, even though concurrent queries randomize the I/O, having a larger block size brings more of the rows into the SGA at once. Having these additional rows in the SGA can benefit you because you *will* be using them.
- ◆ **Clusters.** Depending on the application, you may benefit from clusters. The benefit you receive depends completely on the data access patterns and your application. In general, I don't think that clusters help data warehousing systems, except for very specific applications.
- ◆ **Hash clusters.** Because the data warehouse typically uses table scans to get large amounts of data, a hash cluster will most likely degrade performance. Hash clusters are useful when the queries involve equality statements on the hash key.
- ◆ **Indexes.** Again, because table scans are involved in most queries, indexes may not be particularly useful (although some queries may benefit from the use of indexes). How and when you use indexes depends on your application. If you do use indexes, you may want to take advantage of the FULL hint, which bypasses the index, if you know you will be doing a table scan.
- ◆ **Direct-write sorts.** Direct-write sorts is a new feature that allows you to bypass the SGA when performing sort operations. This feature prevents sorts from taking unnecessary space in the SGA; the result is more SGA space for other operations. If you do a lot of sorting, direct-write sorts may be helpful.
- ◆ **Parallel Query option.** The Oracle Parallel Query option can almost certainly improve the performance of your data warehouse. Parallel queries can be performed on sorts, joins, and table scans to improve the performance of those operations.

Many of these enhancements can help your performance; some may degrade performance. The specific effect on your system will vary, but the information given here is true for most general cases.

Parallel Query Option

Probably the best application to take advantage of the Parallel Query option is a DSS system—and the data warehouse is essentially a DSS system. The design of the Parallel Query option is centered around large queries that access large amounts of data (which is a typical feature of a data warehouse query). Here is a short list of some of the operations that can be parallelized:

- ◆ Sorts
- ◆ Joins
- ◆ Table scans

All these query types are typically used in data warehouse queries. Because these queries are typically of long duration and involve large amounts of disk I/O, you can drastically cut the time it takes to complete these queries by using the Parallel Query option.

Once you set up a test database and develop some test queries (hopefully similar to, if not exactly the same as, production queries), run these queries with various degrees of parallelism to determine optimal performance.

If you will run these queries with other queries in production, you should also test with multiple queries. A tuning parameter set up for use with a single query may not be optimal when used with many queries in parallel.

The Parallel Query option can take advantage of the concurrent nature of disk arrays. By splitting up the queries into many small pieces, you may be able to request data from all the drives in your system simultaneously, optimizing I/O performance.

Oracle Parallel Server

The Oracle Parallel Server option can be beneficial to your operation in two ways:

1. **Performance.** If your system is a candidate for the Parallel Server option, you should see significant performance improvements.
2. **Fault tolerance.** By using the Oracle Parallel Server option, you can keep the system running, even if a computer fails. The data warehouse is required to be available for large periods of time, it typically does not demand the same uptime requirements as OLTP systems.

The Oracle Parallel Server option can enhance the performance of your data warehouse, but only if your system is suited for parallel servers. To see scaleable performance increases, your system must meet the following criteria:

- ◆ **Partitionable data.** The data is partitionable. In other words, the work related to certain tables can be done on one computer while work related to other tables is done on another computer. If all users access the exact same data, your application may not be a good candidate for the Parallel Server option. Although some overlap is fine, the majority of the data should be partitionable.

- ◆ **Many processes.** If you have only a few users and concurrent transactions, you probably are not a good candidate for the Parallel Server option. However, if you use the Parallel Query option, you may use a sufficient number of query server processes to satisfy this requirement.

If these conditions are met, you will probably benefit from using the Oracle Parallel Server option.

NOTE: The Oracle Parallel Server option is not available on all platforms. Consult your system provider or Oracle to determine which platforms support the Oracle Parallel Server option.

Hardware Enhancements

For a data warehouse, there are several hardware enhancements that can improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements, assuming that you are not I/O bound. The speed of CPUs is constantly being improved as are new and better cache designs.

For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before you purchase an additional processor of the same type and speed, however, consider upgrading to a faster processor. For example, upgrading from a 66 MHz processor to a 133 MHz processor may provide more benefit than purchasing an additional 66 MHz CPU—with the added benefit that you now have the option of adding more 133 MHz CPUs. Because of the complexity and run time required by these queries, you can benefit from more and faster CPUs.

SMP and MPP computers provide scaleable CPU performance enhancements at a fraction of the cost of another computer. When upgrading your processors or adding additional processors, remember that your I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when you add processors.

I/O Enhancements

You can enhance I/O by adding disk drives or purchasing a hardware disk array. The data warehouse can benefit from the disk striping available in both hardware and software disk arrays. Using Oracle data file striping can also help the performance of your data warehouse.

If your system performs only one query at a time and you are not taking advantage of the Oracle Parallel Query option, you may not see a benefit from a hardware or software disk array. In this specific case, I do not recommend OS or hardware striping; you should use traditional Oracle striping. Because you are executing only one query at a time without using the Parallel Query option, the I/Os to the data files are purely sequential on the table scans. This scenario is somewhat rare; any variance from “pure table scans” results in degraded performance.

Hardware and software disk arrays have the added benefit of optional fault tolerance. As described in Chapter 15, “Disk Arrays,” each of the fault-tolerant RAID levels has its advantages and disadvantages. You should first choose the correct fault tolerance for your needs and then make sure that you have sufficient I/O capabilities to achieve the required performance level. If you use fault tolerance, you will most likely have to increase the number of disk drives in your system.

TIP: Because data warehousing systems primarily involve a majority of read operations and fewer writes on some data, you can take advantage of the fault tolerance and read performance of RAID-5. Make sure that write activity is small and remember that there may be significant write activity in the temporary tables.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If your queries often perform table scans, you may see good improved performance with disk controllers that take advantage of read-ahead features.

Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the data warehouse where you are performing DSS queries, it is likely that you will need that data soon; if you do, it will be available very quickly.

CAUTION: It is important to make sure that any controller you use with a write cache is protected against a power failure. Some disk array controllers on the market today (such as the Compaq disk array controllers) offer a battery-backed, mirrored memory cache that protects your data. Remember: Once Oracle believes that data has been written to the drive, that data had better be there.

Enhancements to the I/O subsystem almost always help in a data warehouse environment because large amounts of data are accessed. Be sure that you have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

Network Enhancements

Network performance may or may not be an issue in your data warehouse. Depending on the load generated during loading phases, you may see some degradation caused by a slow network. If this is the case, you can make several enhancements.

Segmenting the network can improve performance. This is particularly useful when data is loaded from several different machines. It may be necessary to reduce other traffic on the segments connecting the data warehouse to the data sources.

You may be able to take advantage of new networking hardware such as 100 megabit Ethernet and fiber-optic networks. When you increase the bandwidth of the network, you can transfer more data. Check your network periodically to see that you do not have excessive collisions or bandwidth problems. Any problems with the network can be easily solved by adding more or faster hardware.

Review of Enhancement Options

By taking advantage of some of the performance enhancement features available, you may see significant performance improvement. I have worked with the Oracle Parallel Query option and I am very impressed with the performance gains possible. Data warehousing systems are particularly well suited for the parallelizing of queries.

Using the Parallel Query option in conjunction with a disk array can be beneficial to your system's performance. By combining these enhancements with other tuning options (such as a large block size and multiblock reads), you are on your way to having a well-tuned data warehouse.

Performance Verification

Once you design and build your system, you must have some way of verifying performance. Verifying the performance of a data warehousing environment is much easier than with OLTP systems because you do not have to simulate large numbers of users.

The simplest method is to test the system using production queries on a test database. By saving the test database and using the same queries on the same data, you can make direct comparisons when you change tuning parameters or upgrade the system.

A good documented test plan allows you not only to verify system performance but to try new things, such as the Parallel Query option, direct-write sorts, or some other new feature you may want to implement in the future. The following sections look at what should be tested in the RDBMS and the operating system.

What To Test in the RDBMS

You should test the RDBMS keeping functionality, load testing, and performance in mind. Make sure that the system will function as specified. Try to use a test database that is configured and sized similar to the production database.

Once you build an effective load test with repeatable results, start changing some of the parameters described in Part II of this book and earlier in this chapter. By changing only one parameter at a time, you can easily see which parameters improve performance and by how much.

Before changing anything, set your expectations for the changes you are going to make. If the results don't turn out as expected, try to understand why they didn't. Be sure to log the results so that you have a reference of what changes were effective and what changes were not.

What To Test in the OS

For the most part, the OS is tested with the RDBMS. If you have to change some specific OS parameter to increase a limitation, you usually don't have to retest the performance. Any limit change is usually associated with an RDBMS change, and the two can be tested together.

Other OS changes such as feature enhancements should be tested with the RDBMS, but you should more closely watch and analyze these results. You should carefully analyze things such as scheduler changes and cache affinity that may have an adverse affect on the system to verify that performance has not degraded.

As stated earlier in this book, the OS is primarily a vehicle for the RDBMS to use. The primary goal in tuning the server OS is to reduce overhead and optimize the I/O, memory, and networking subsystems.

Benchmarks

Standardized benchmarks are a good way to judge how well a system is performing and to use as a comparison between different hardware platforms. If they are available to you, standardized benchmarks are also a good way to analyze the performance of your particular platform.

Although a data warehouse is not specifically a DSS system, it does involve many decision support queries. Because this is so, you may find that a standardized benchmark (such as the TPC-D benchmark) offers some guidance and information. This information may be useful in helping you determine the type of hardware you should purchase.

Using a standardized benchmark (such as the TPC-D benchmark) is not a good way to tune your system. You should tune each system individually, based on its own characteristics. Examine the Full Disclosure Report (FDR) submitted by the test sponsor to see a breakdown of performance by query type. Look at the performance for the particular query types you use in your operation. Doing so may give you some indication of how well the testing configuration would work in your environment.

A system tuned for a benchmark such as the TPC-D benchmark may provide useful tuning hints. Because the primary goal of a TPC benchmark is for the sponsor to get the best possible performance, you can see how the sponsors have optimally tuned their systems.

Because an official TPC-D benchmark using Oracle is usually submitted by the hardware manufacturers, OS vendors, and Oracle, you can be assured that the system has been tuned as optimally as possible. If you can, obtain a Full Disclosure Report from the TPC (on the Web at this URL: <http://www.tpc.org>); look at the way the system was designed and tuned.

Of course, the best benchmark to use on your data warehousing system is one of your own production decision support queries. This benchmark provides you with all the information you need to judge the amount and size of the hardware required. You can also use this test to determine whether tuning or hardware changes have improved performance and by how much.

It is also important to benchmark and test the loading and data-manipulation phases used in generating the data warehouse's data. Because this process can be quite complex and time consuming, do not neglect the performance of this phase or operation. If you combine data from several sources to generate new data, you may be spending several hours a day adding this data.

You can reduce the time it takes to generate this data by properly tuning memory and I/O for this task. Depending on the requirements of the system, it may be more crucial to tune the load phase than the query phase of the daily operations.

Summary

As the price of computer hardware—especially disk drives—comes down every year, the idea of a data warehouse becomes increasingly more feasible. The amount of hardware that a few short years ago would have cost millions of dollars can now be obtained for much less. The performance of this hardware is also increasing at incredible rates. Systems that were considered minicomputers a few years ago are now being replaced by PC servers with twice the capacity at half the cost.

The ongoing reduction in cost and increase in performance will promote the trend of larger and larger databases with better information retrieval. The goal of the data warehouse is to take information from production databases, legacy data, and outside sources and use this information to better your business. As hardware gets faster and faster at less cost, this trend will continue; new applications and ways of looking at your business data will be developed. It's very exciting to look at the way the RDBMS industry has grown in the last few years. It will be exciting to see where it goes in the next few years.

This chapter looked at the characteristics of the data warehouse as a business model and from the data access perspective. You can use this information to determine how to configure the system to take advantage of the data access patterns. If you understand how the system operates and what affects the performance of the system, you can use this information to design a system that has well-balanced I/Os and that can take full advantage of the computer's CPU power.

Chapter 20

BLOB System

Oracle can store Binary Large Objects (BLOBs) using the RAW or LONG RAW data type available from Oracle. BLOBs are used to store data that cannot be stored in a textual form. Such data can have any of the following forms:

- ◆ **Images.** These images may have any of several formats, including GIF and JPEG.
- ◆ **Video.** These types of images may be in MPEG or some other format.
- ◆ **Audio.** Audio data can be stored in a variety of formats, including PCM, ADPCM, DSP, GSM, and so on.
- ◆ **Documents.** Scanned documents can be stored as a BLOB.

These and many other types of data can be stored in an Oracle database using the RAW or LONG RAW data type. These data types permit you to store any type of data that does not fit into the typical data types. This allows you immense flexibility in the type and amount of data you can store in a database. With the increased popularity of multimedia applications and data, the effective storage of such data is becoming more important.

Characteristics of BLOBs

Binary large data is a term used to describe any type of binary data that is significant in size and stored on a computer. Although BLOBs do not have to be stored in a database, for the purposes of this book, only data stored in an Oracle database is addressed.

BLOBs stored in an Oracle database usually exhibit the following characteristics:

- ◆ **Significant size.** Each record is significant in size. Although the size depends on your particular database and application, with BLOBs, the objects can be 1M in size or larger.
- ◆ **Binary data.** BLOB data is binary by definition.
- ◆ **Strict response time criteria.** Because the data can consist of sources that require continuous data feeds (such as video and audio), the response-time criteria may be strict. When numeric data is momentarily interrupted, the user hardly notices, but when an audio or video stream is interrupted, it is quite noticeable.
- ◆ **Compression.** Because the data is binary and large, it may or may not be stored in a compressed state. This depends on the database and the application.

These are the general attributes of binary large objects. The way this data is accessed is quite different from other types of applications described in the preceding chapters. The following section looks at the data access patterns for BLOBs.

Data Access Patterns

The data access patterns for a system used for BLOB storage are unique. One or more tables have records of a megabyte or more. When this data is accessed, it is a continuous data access to a single record. In the other types of database applications you have seen, it is unlikely that a single record is larger than a data block. With BLOBs, it is certain that a single record will span many data blocks.

System Load

The load you see when accessing BLOBs is different than the typical load you see with other types of RDBMS systems. With BLOBs, the load tends to be more on the I/O subsystem and less on the system processors.

In a typical OLTP, batch, or DSS system, the system processors are kept busy determining where to look for the next piece of data, completing some other function such as a sort, or performing aggregate functions. It is also normal for the I/O subsystem to spend most of its time seeking to new data rather than waiting for sequential data to be retrieved. In fact, in most type of RDBMS operations, it is normal for more time to be spent seeking than retrieving the data.

In a system used to access BLOB data, it is not unusual for more time to be spent retrieving the data than seeking to it. This reversal occurs because of the large amount of data associated with a typical BLOB record. It is not unusual for the BLOB system to spend a lot of time in I/O activities; therefore, it is less sensitive to the CPU horsepower of the system.

Because the data accesses to these large records are sequential, simply adding more disk drives to the system may not solve the problem. However, because most of these accesses occur simultaneously with other disk accesses, many of the concepts discussed in previous chapters still apply. A disk array can still be very beneficial.

Other tuning concepts also apply but may vary slightly from previous examples. The primary emphasis in accessing BLOB data is the I/O subsystem. The performance of your system is based primarily on the performance of the I/O subsystem.

Following is a list of the load characteristics of a BLOB system:

- ◆ **Relatively few processes on the system.** For the BLOB system, the Oracle Parallel Query option is not of any benefit.
- ◆ **Extremely high network traffic.** The primary function of this type of system is to provide some sort of data stream. Typically, this function is to service remote clients.
- ◆ **Heavy I/O usage.** The BLOB system depends heavily on the I/O subsystem. Although the accesses are sequential, multiple simultaneous accesses make this I/O random. By taking advantage of multiblock reads, sequential access is done as much as possible.
- ◆ **Very little redo log usage during access, heavy usage during data loading.** These types of systems are primarily used for data retrieval; however, adding data to the system can significantly affect the redo log.
- ◆ **Very little or no use of rollback segments.** Again, because BLOB functions consist of very little update activity, the rollback segments are hardly used. However, if data loading is done with `INSERT` statements instead of with the direct path loader, rollback segments may be affected.
- ◆ **Moderate to heavy use of memory.** Memory is used not only for the SGA but for each of the server processes that may be performing additional functions.

You can use these characteristics to help design and tune your BLOB system for optimal performance. The first step in this design process is to set goals for what you want to achieve.

Goals

The goal in tuning your system for BLOB access is to achieve a system that has certain characteristics. Here is a list of the characteristics of an optimally tuned BLOB system:

- ◆ **The system shows minimal indications of being disk bound.** If your system is disk bound, you should add more or faster disks. At some point, the BLOB system may overwhelm the I/O subsystem. You may have to add computers to help with the load.

- ◆ **Memory is sufficient.** If the machine pages or swaps, performance is severely degraded. The best solution is to add more memory; if that is not possible, reduce the size of the SGA or the number of data streams until the system no longer pages or swaps.
- ◆ **There is sufficient space in the shared pool.** I don't think that the shared pool will be much of a problem with this type of data access. However, check your system to verify that fact.
- ◆ **The system meets any additional requirements you have.** One criteria that may be important is that the system must support a continual data flow. If your data involves audio or video information, any type of delay will be very noticeable.

By setting goals for how you expect the system to perform, you can determine whether you are successful. You can also determine earlier whether you will be able to achieve your specified goals.

Review of BLOB System Characteristics

By analyzing the characteristics of a system designed for BLOB access and setting goals you want the system to achieve, you can obtain a lot of data with which you can build the system. The design of the system is largely determined by how data is accessed and how many and what type of queries are specified.

You have seen that the system used for BLOB access is generally an I/O-bound system. These I/Os are sequential in nature, but multiple accesses break them into a more random pattern. By configuring the system and taking advantage of multiblock reads, you can achieve a high level of I/O performance.

Relate the information in the next part of this chapter to your particular configuration. Look for similarities and differences between what is described here and what you have observed about your system and decide how you can benefit from the tuning guidelines presented here. Although each system is different, many of the concepts remain the same.

You should base your system design on what you know about how your system operates and accesses data. Spend the time up front to carefully examine how the system needs to run so that you can determine the design of the system.

With this type of system, some design choices can drastically affect the overall performance of the system. If you use compression, you can choose to compress/decompress the data on the client where CPU resources are more available. The amount and size of the data may help you determine how to divide the data among different disks.

Because there are some inherent limitations to the number of data streams a single system can support, you may want to partition the data into multiple systems. If possible, partition the data to spread the data access equally among the machines.

How well the system performs is often reflected in the amount of effort you put into the original design of the system. It is difficult to make up for a poor design with tuning.

Design Considerations

By looking at the data access patterns, you should have a good idea of how the system will operate. Before looking at the design of the system, here is a review of a few of the concepts introduced in earlier chapters:

- ◆ **I/O is typically the limiting factor in the system.** You can do only a fixed number of random I/Os per second per disk drive (refer to Chapter 14, “Advanced Disk I/O Concepts”).
- ◆ **I/Os can be reduced by caching data blocks in the SGA.** If the data you want to access is already in the SGA, a disk I/O is not required.
- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out among many drives, you increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

All these factors contribute to the optimal data layout of the system. When designing a system for BLOB access, the physical layout is of primary importance. It is also important to be aware of the application and how the data will be accessed and compressed or decompressed.

Because large amounts of binary data are accessed in a BLOB system, compression can be very important. If high-speed clients are used, compressing and decompressing the data on the clients may be effective. If video or audio data is used, it can help to know that many new applications can access this compressed data directly. With this type of environment, it is advantageous to move the data as quickly as possible to the client machines and let them do the processing.

Physical Data Layout

This section looks at how the data in a BLOB system should be configured on the physical disks. First, it looks at how to lay out the data on traditional disks; then it looks at disk arrays. I recommend using disk arrays if at all possible; the ease of use and performance benefits are worth the cost of the array.

The main goal in designing the physical data layout is to balance the I/O across all the disks that are randomly accessed and to isolate the sequential I/O. Depending on your application and how often you insert new data into the system (or whether you use the system simply for data retrieval), the redo logs may or may not be significant.

If data is updated or inserted on a regular basis, the redo log files should be separated. If little or no update/insert activity occurs, you can use a minimal redo log configuration. In a BLOB system, you know that the majority (if not all) of the data files are accessed in a random fashion but may take advantage of multiblock reads. The individual object requests are done with sequential requests, but because many data streams will be active, the sequential access is randomized.

In other chapters, I recommended that more, smaller disk drives are more advantageous than a few larger disk drives. For system that access BLOBs, I make an exception. It is still important to take advantage of as many disks as possible, but if a 4.3 gigabyte disk has a significantly faster data access time than a 2.1 gigabyte disk, I recommend the faster disk. I reverse my usual recommendation because much of the data access is done using sequential I/O of the size of the multiblock read.

Traditional Disks

The layout for a system used for BLOB data is somewhat more complicated than most configurations. A minimal configuration should look something like this:

Element (# of Disks)	Comments
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries.
Redo log (0)	<i>Little or no insert/update activity.</i> Because very little update activity is involved with this system, the log files are not active and are not a bottleneck. The disk drives normally allocated as redo log drives can be better used for data files. The redo log files can reside on the same disk as the system.
Redo log (2+)	<i>Some insert/update activity.</i> If online updates are done to the system, you must separate the redo log files for both protection and performance. Whether you need to add separate redo log drives or not depends on the extent of the updates.

<i>Element (# of Disks)</i>	<i>Comments</i>
Archive logs (0)	<i>Little or no insert/update activity.</i> As with the redo log files, archiving is minimal and not necessary when there is little update activity. The archive log files can also reside on the system disk.
Archive logs (1+)	<i>Some insert/update activity.</i> As with the redo log files, if updates are done on this system, you must provide for the sequential nature of the archive log files.
Data files (1+)	The number of disks you need for data is determined by the amount of random I/O the requests for data generates and the size of the database.
Index files (0)	This type of system typically uses very small indexes because the large amount of data in the system is usually made up of a small number of very large records. No separate index disk drives are needed; these indexes can reside on the same disk drives as the data.

The data files should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

As you have seen in previous chapters, the data can be striped across the disks using Oracle or RAID striping or a combination of the two. Because the disk array provides excellent performance and is easy to use, when you use a disk array, the task of distributing I/Os can be greatly simplified.

Disk Arrays

For a system that accesses BLOBs, the layout using a disk array is much simpler than it is with traditional disk drives. A minimal configuration should look something like this:

<i>Element (# of Volumes)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, you should mirror this disk for additional fault tolerance.
Redo log (0)	<i>Little or no insert/update activity.</i> Because there is very little update activity involved with this system, the log files are not active and are not a bottleneck. The disk drives normally allocated for the redo log volumes can be better used for data files. The redo log files can reside on the same disk as the system.

continues

<i>Element (# of Volumes)</i>	<i>Comments</i>
Redo log (1+)	<i>Some insert/update activity.</i> If online updates are done to the system, you must separate the redo log files for both protection and performance. Whether you need to allocate an additional logical volume for redo log files depends on the extent of the updates. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance degrades during archiving because the sequential nature of the log writes is disrupted.
Archive logs (0)	<i>Little or no insert/update activity.</i> As with the redo log files, archiving is minimal and not necessary when there is little update activity. The archive log files can also reside on the system disk.
Archive logs (1)	<i>Some insert/update activity.</i> The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary.
Data and index (1+)	Because you always get concurrent access to disks with a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O the requests for data generates and the size of the database.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

When you use a disk array, the data is automatically striped across all the disk drives; therefore, it is necessary to create only one tablespace and table for all your data. You do not even have to put indexes into another tablespace—although I recommend doing so for other reasons (such as monitoring and maintenance).

With traditional disks, it is difficult to manage hundreds of data files and disks; with a disk array, you can manage hundreds of disks with just a few data files. Of course, Oracle has a 2 gigabyte limitation on the size of a data file, but this is easily resolved by creating a data file for every 2 gigabytes of space you need. The data files can all reside on the same disk array volume.

If you use a disk array, many of the management tasks and load balancing tasks are greatly simplified. With the disk array, you also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

I recommend that you use a disk array if possible. Software striping is fine, but if your system is under heavy loads (as is typical in a system used for BLOB data), you can achieve better performance by offloading the striping overhead to a hardware RAID controller.

Hardware Considerations

When choosing hardware to use for a system used for BLOB data, consider these factors:

- ◆ **Low user load.** Not many concurrent processes/threads simultaneously access the system.
- ◆ **High I/O load.** I/Os are concurrent and heavy; they are mostly random I/O with a size of multiblock reads. The I/O pattern requested by an individual process is sequential, but because there are multiple data streams, the data access becomes more random.
- ◆ **Extremely high network traffic.** Because the idea of BLOB systems is to take large amounts of data and quickly transfer that data across the network, the amount of traffic is usually very high.

Because this type of data access does not typically involve heavy CPU activity, you may not see any benefit from multiple CPUs. It is much more important to have a system that can handle extreme amounts of I/O and network processing. This is not to say that the CPU activity will not be high, but the BLOB system is not as CPU bound as is a typical OLTP or DSS system.

With this type of access to the disks, you can benefit from a disk array. I prefer hardware striping to OS striping because it does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is adequate.

Because network traffic is very high, you must have a sufficient network bandwidth to handle the load. A high-speed network solution such as 100Base-T or fiber optics can benefit the overall performance of the system.

Tuning Considerations

The system used for BLOB access is designed to give optimal access to large amounts of data as quickly as possible. It is also frequently a requirement that the data streams be continuous for both video and audio data.

You may have to tune both Oracle and the server operating system. The following sections look first at Oracle and then at the server operating system.

Oracle Tuning

Carefully analyze these things to determine whether adjustment to these parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** Because such large amounts of data are accessed either frequently or infrequently, it is hard to determine whether the SGA is effective for caching data. Unless the same BLOB data is accessed by several users, the SGA is typically not useful for data caching.
- ◆ **Library cache.** Check the V\$LIBRARYCACHE table, which contains statistics about how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You can reduce the library cache misses by increasing the amount of memory available for the library cache. Do this by increasing the Oracle tunable parameter SHARED_POOL_SIZE. Try increasing the size of the shared pool by 10 percent and monitor it again. If this is not sufficient, increase the size of the shared pool by another 10 percent and continue in this fashion.
- ◆ **Multiblock reads.** Because the BLOB data is accessed sequentially, the size of the multiblock read is important. The number of blocks read in a multiblock read is specified by the Oracle initialization parameter DB_FILE_MULTIBLOCK_READ_COUNT. This value, multiplied by the DB_BLOCK_SIZE parameter, results in the size of the I/O. A good value for this initialization parameter is 64K or larger; try setting this parameter as high as Oracle allows. The maximum size is platform dependent; check your Oracle documentation for this value.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, look at the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback segments.** Unless you do a significant amount of updates, rollback contention and the number of rollback segments is usually not important in BLOB systems.
- ◆ **Latch contention.** Unless you do a significant amount of updates, latches are not an issue in a BLOB system.
- ◆ **Checkpoints.** Unless the number of updates is significant, the checkpoint processes is usually not an issue for this type of system.
- ◆ **Archiving.** As with the other redo log-related parameters, because the BLOB system is primarily used for reads, archiving performance is usually not a problem.

Pay particular attention to these areas when tuning a system for BLOB access. Probably the area that requires the most attention is the I/O subsystem. By optimizing the I/O subsystem, you will see performance increases.

Server OS Tuning

You may have to tune the server OS to provide for optimal I/O performance. Some of the things you may have to tune in the server OS are listed here; remember that some OSes may not require any tuning in these areas:

- ◆ **Memory.** Tune the system to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes.
- ◆ **Memory enhancements.** Take advantage of 4M pages and ISM, if they are available. Both features can improve Oracle performance in a BLOB environment.
- ◆ **I/O.** Tune I/O to allow for optimal performance and use of AIO.
- ◆ **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In a BLOB system, allowing a process to run to completion (that is, so that it is not preempted) is beneficial.
- ◆ **Cache affinity.** You may see some benefits from cache affinity because the amount of data is large and the number of processes is few.

The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, you can enhance Oracle performance.

Enhancements

You may want to make some additional enhancements to improve the performance of your system. Listed here are some of the items that can offer performance improvement; also listed are a few items that do not improve performance. Each item is accompanied by an explanation.

- ◆ **Block size.** In a BLOB system, increasing the database block size greatly improves performance. Because many of the queries access the database in a sequential manner, even though concurrent queries randomize the I/O, having a larger block size brings more of the rows into the SGA at once. Having these additional rows in the SGA can benefit you because you *will* be using them.
- ◆ **Clusters and hash clusters.** I don't believe that clusters or hash clusters are of any benefit in this type of system.
- ◆ **Indexes.** Because the size of the data is very large, the database is full of data that represents a relatively small number of rows. Using an index may be beneficial in this case because a table scan involves lots of unused data.
- ◆ **Parallel Query option.** The Oracle Parallel Query option is of no use to you in this type of system.

Some of these enhancements can help your performance. The specific effect on your system will vary, but the information given here is true for most general cases.

Hardware Enhancements

In a BLOB system, several hardware enhancements can help you improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements—assuming that your application is causing the system to become CPU bound.

For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before you purchase an additional processor of the same type and speed, consider upgrading to a faster processor. In this type of environment, both the addition of faster processors and more processors benefit the overall performance of the system.

I/O Enhancements

You can enhance I/O by adding disk drives or purchasing a hardware disk array. Systems that access BLOB data can benefit from the disk striping available in both hardware and software disk arrays. Using Oracle data file striping can also help the performance of this type of system.

Hardware and software disk arrays have the added benefit of optional fault tolerance. As described in Chapter 15, “Disk Arrays,” each of the fault-tolerant RAID levels has its advantages and disadvantages. You should first choose the correct fault tolerance for your needs and then make sure that you have sufficient I/O capabilities to achieve the required performance level. If you use fault tolerance, you will most likely have to increase the number of disk drives in your system.

TIP: Because BLOB systems involve a majority of read operations and very few writes, you can take advantage of the fault tolerance and read performance of RAID-5. You should make sure that write activity is small.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If your queries often perform table scans, you may see good improved performance with disk controllers that take advantage of read-ahead features.

Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the BLOB system, you *will* need that data soon; if it is in the cache, it will be available very quickly.

CAUTION: It is important to make sure that any controller you use with a write cache is protected against a power failure. Some disk array controllers on the market today (such as the Compaq disk array controllers) offer a battery-backed, mirrored memory cache that protects your data. Remember: Once Oracle believes that data has been written to the drive, that data had better be there.

Enhancements to the I/O subsystem make all the difference in the world to a BLOB environment because large amounts of data are being continually read. Be sure that you have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

Network Enhancements

Network performance can be quite a bottleneck for this type of system. Because large amounts of data are transferred across the wire, a high-speed link is imperative.

You may be able to take advantage of the new 100Base-T technology or benefit from a fiber-optics link. The faster the network link, the faster the data is transferred. You may find it beneficial to subnet your networks between the high-access systems. Try to use the latest technology (such as 100 megabit Ethernet) to provide for future expansion.

Review of Enhancement Options

By taking advantage of some of the performance enhancement features available to you, you may see significant performance improvement. In this type of system, I/O is of the highest importance. By taking advantage of large block sizes and multiblock reads, you can improve data access performance. Use these design considerations in conjunction with a well-designed application and you will have a good-performing system.

Performance Verification

To verify that changes you have made have positively affected performance, you must develop some sort of test plan. The test plan should consist of an application that is very similar to the application that will be used to access this data. The closer your tests are to the actual application, the more accurate the performance verification will be.

A good documented test plan allows you not only to verify system performance but to try new tuning parameters or new features you may want to implement in the future. The following sections look at what should be tested in the RDBMS and the operating system.

What To Test in the RDBMS

The test should be designed to access the data in a manner consistent with the data access patterns seen with the actual application. If the application involves video streams across the network, verify this data flow in a manner as similar to production as possible. If strict data throughput criteria are involved, verify that you can meet this criteria by using a database that is close in size to the production database.

Once you build an effective load test with repeatable results, start changing some of the parameters described in Part II of this book and earlier in this chapter. By changing only one parameter at a time, you can easily see which changes affect performance.

Before changing anything, set your expectations for the changes you are going to make. If the results don't turn out as expected, try to understand why they didn't. Be sure to log the results so that you have a reference of what changes were effective and what changes were not.

What To Test in the OS

For the most part, the OS is tested with the RDBMS. If you have to change some specific OS parameter to increase a limitation, you usually don't have to retest the performance. Any limit change is usually associated with an RDBMS change, and the two can be tested together.

Other OS changes such as feature enhancements should be tested with the RDBMS, but you should more closely watch and analyze these results. You should carefully analyze things such as scheduler changes and cache affinity that may have an adverse affect on the system to verify that performance has not degraded.

As stated earlier in this book, the OS is primarily a vehicle for the RDBMS to use. The primary goal in tuning the server OS is to reduce overhead and optimize the I/O, memory, and networking subsystems.

Benchmarks

Standardized benchmarks are a good way to judge how well a system is performing and to compare different hardware platforms. If they are available to you, standardized benchmarks are also a good way to analyze the performance of your particular platform.

Although you can benchmark an OLTP system with the TPC-C benchmark and a decision support system with the TPC-D benchmark, there are no standardized benchmarks currently used to demonstrate the performance of BLOB access.

The best way for you to benchmark your system is with the production application. If possible, modify this application to time certain events, such as the following:

- ◆ Start time: when the request for data is submitted.
- ◆ Response time 1: the time when the data starts arriving at the client.
- ◆ Response time 2: the time when all the data has finished arriving at the client.

In this manner, you can quantitatively measure the performance of your system. If you make any changes, you can run them against this data and compare the results. Try to create a benchmark that has a variable load so that you can push the system to its limits to gather maximum load information.

Summary

This chapter looked at a system designed to access BLOB data. Specifically, it looked at the methods and data patterns generated when BLOB data is accessed. By looking at these data access patterns, you discovered some ways to design a system to take advantage of this information.

By taking advantage of large block sizes and multiblock reads, you can increase the performance of access to this data. The I/O subsystem is of critical importance in this type of system and should be sufficient to handle the required workload.

This type of system is frequently used to view the same data over and over again. This is advantageous because it allows you to try new tuning parameters and be innovative. By having essentially the same job to run, you can quantitatively measure the results of any changes and determine whether performance has changed—either positively or negatively. By logging the results of these tests, you gather valuable data that can help determine which changes should be tried in the future.

Chapter 21

The Oracle Parallel Server System

The Oracle parallel server system is an option to the Oracle7 Server; it allows you to link several computers together to form a cluster. A *cluster* is a group of computers that work together to provide a unified service. In the case of the Oracle parallel server cluster, these systems are linked together to form one large RDBMS system.

Primary among the reasons many people choose the Oracle parallel server system is fault tolerance, but many also use the parallel server for the increased performance. But an Oracle parallel server system can only achieve good performance scalability if you properly design and configure the system.

The Oracle parallel server system is made up of several nodes or cluster members, each of which runs an Oracle instance. Each of these instances accesses the same database through a shared disk system. Because the redo log files and rollback segments also reside on the shared disk system, it is possible for a node to perform instance recovery on another cluster member in the event of a failure. If a node fails, the other nodes in the cluster are not affected.

Not all applications benefit from the use of the Oracle Parallel Server option. Applications that are particularly well suited for the Parallel Server option have the following attributes:

- ◆ **Very little or no update activity.** The lack of activity reduces the amount of locking involved and allows for a high level of concurrency.
- ◆ **Applications are partitionable.** Applications can be partitioned so that users who update a certain table or set of tables use a single server, while other users who update unrelated tables can use another server. Partitioning the application cuts down on the contention between servers.
- ◆ **Very large applications.** Large applications that usually do not see a high number of cache hits can benefit from the Parallel Server option. The increased CPU power of the cluster shows increased performance and because the data is not likely in cache, this indicates that a PCM lock is not likely on that data already.
- ◆ **Query-intensive applications.** Applications with many large queries can take advantage of both the Parallel Server option and the Parallel Query option.

This chapter presents some of the techniques involved in partitioning data and users to achieve the best performance possible from the Oracle parallel server system. Before you look at the specifics involved in configuring the parallel server system, the following section reviews the architecture of this option.

Oracle Parallel Server Architecture

With the Oracle Parallel Server option, you can cluster or join several systems together to form one larger RDBMS system. Because of the addition of parallel server nodes, the Oracle parallel server system offers fault-tolerance features as well as performance increases.

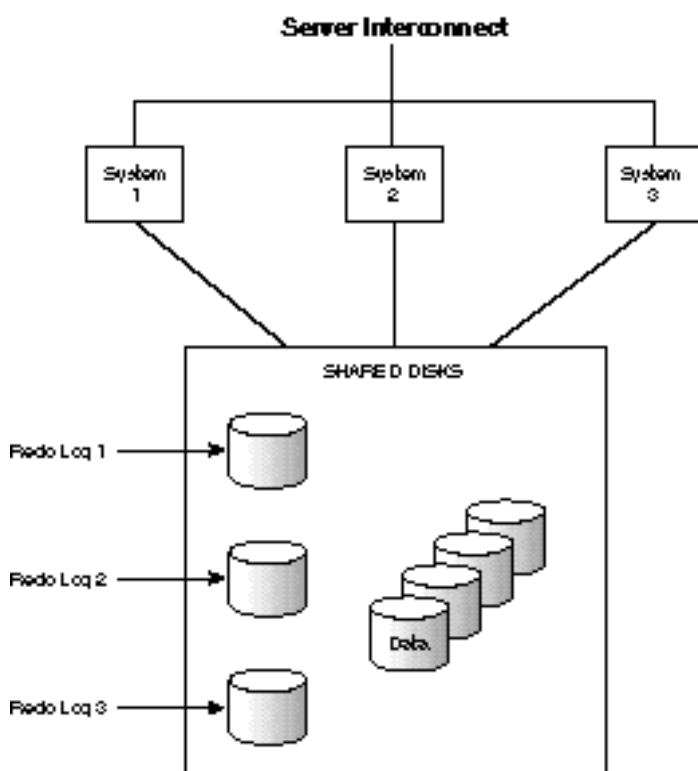
The Oracle parallel server system is made up of several nodes that access the same database through a shared disk system. Because the redo log files and rollback segments also reside on this shared disk system, one node can perform instance recovery on another cluster member in the event of a failure. If a node fails, the other nodes in the cluster are not affected.

This fault tolerance feature eliminates downtime in the event of a node failure. If the system is properly designed, you can obtain performance increases by adding additional nodes to the cluster.

To take advantage of the scalability of the Oracle parallel server system, you should understand how the system operates. The parallel server system allows multiple Oracle instances to share the same database through a *parallel cache management* system and a shared disk I/O system (see Figure 21.1).

Figure 21.1

The Oracle parallel server system.



The Oracle Parallel Server option uses a sophisticated locking mechanism in conjunction with a shared disk subsystem to allow multiple instances to access the same data. Communication between the computers is done through a *server interconnect*, which usually consists of one or more high-speed network devices that communicate at a very low level. Using the traditional network stack does not provide the performance required for a cluster interconnect.

The server interconnect provides two functions: It is used to communicate locking information and to provide a system heartbeat. The system heartbeat communicates to other systems in the cluster that the system is still operational. If the heartbeat message does not arrive, other servers in the cluster assume that the system is nonfunctional and roll back transactions that have not been committed.

Locking is performed through a process called the Distributed Lock Manager (DLM). The DLM is responsible for locking data that is being modified so that it cannot be modified in another instance. Locking ensures data integrity across the entire cluster. A data block or group

of blocks is locked until another instance needs that data. The DLM provides the following functionality:

- ◆ Keeps track of resource ownership
- ◆ Queues requests for PCM locks
- ◆ Notifies an instance that a resource is requested
- ◆ Notifies the requesting instance that the resource has been acquired

Because the locks are held until needed by another instance, if you can partition your users so that users accessing data in a particular table all use the same instance to access that data, you will reduce lock contention. Performance can be enhanced by carefully partitioning the data and the users. If you partition the data into update-intensive and read-intensive tables, you will also benefit.

TIP: Because the PCM locks are held until the data covered by those locks is requested by another instance, partitioning your data to reduce the amount of data that is accessed by different nodes in the cluster can greatly improve performance.

At instance startup, a number of Parallel Cache Management (PCM) locks are created. PCM locks are used to lock data blocks being accessed within each instance to guarantee that multiple instances do not alter the same data. These PCM locks may cover hundreds, thousands, or even millions of data blocks, depending on your configuration.

Locks can be acquired in one of two modes:

<i>Mode</i>	<i>Description</i>
Exclusive Lock mode	When a lock is acquired in Exclusive mode, the instance is allowed to update the blocks covered by the lock. If those blocks are already in use, the DLM requests the other instance to release the lock.
Read Lock mode	When a lock is acquired in Read mode, the instance is assured that during the time the lock is active, the data will not change. It is possible for multiple instances to have read locks on the same data blocks.

PCM locks can be used to lock data blocks for reading or for updating. If a PCM lock is used as a read lock, other instances can acquire read locks on the same data blocks. Only when updating a block must an exclusive lock be acquired.

TIP: Because PCM locks are used to lock data to guarantee data consistency between nodes, you can eliminate the need for PCM locks by taking advantage of read-only tablespaces. Because read-only tablespaces cannot be updated, Oracle does not have to allocate PCM locks for those tables.

When an instance wants to update a data block, it must acquire an exclusive lock on that block. If another instance has a lock that includes that block, the instance holding the lock must write the data out to the shared disk before the lock can be released, a process known as *pinging*.

Because PCM locks are allocated to data files, you have some flexibility over the configuration of the locks. A PCM lock can lock one or more data blocks, depending on the number of PCM locks allocated to the data file and the size of the data file. Because there is an inherent overhead associated with PCM locks, it is not beneficial to overconfigure the locks.

Because a PCM lock covers more than one block, you may unnecessarily be pinging if you want a block that is not being used by the other instance. This is why, with very active tables, you may want to add more locks; with less-active tables, you can reduce the number of locks.

The Parallel Server option can be quite effective if your application is partitionable. If all the users in your system must access the same data, however, the Parallel Server option may not be for you. But if you can partition your workload into divisions based on table access or if you need a fault-tolerant configuration, consider the Parallel Server option.

If you use the Parallel Server option, take special care to properly configure the system. By designing the system properly, you can take maximum advantage of the parallel server features.

Design Considerations

The primary way to optimize performance with a parallel server cluster is to properly configure the system. By properly configuring the system to avoid contention, you can see very good performance by adding nodes. With the Parallel Server option more than any other Oracle feature, it is important to properly design the system for performance.

In a poorly designed parallel server cluster, you may see no performance improvement by adding nodes; you may even see a degradation in performance. By understanding the basic principles of the Oracle parallel server and setting good design goals, you can achieve amazing performance from the cluster.

Design Goals

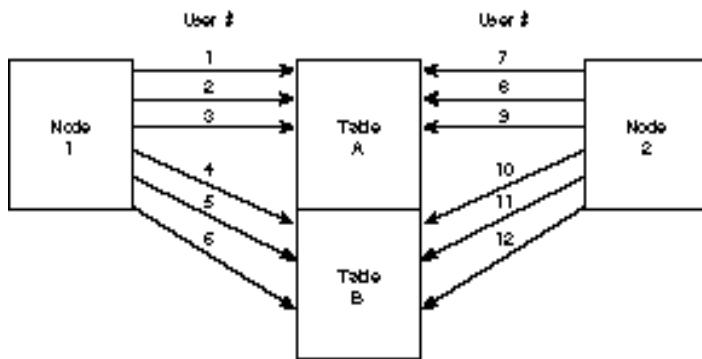
The goals in designing a parallel server system are fairly straightforward. Your primary goals are to reduce contention between servers and to reduce the number of PCM locks that must be obtained and released. By reducing contention on specific data areas, you can minimize the use of locks and therefore increase performance.

Assuming that the I/O subsystem can handle the required shared throughput, the only thing that can slow you down is contention between systems. If this contention is reduced or eliminated, the system will show scaleable performance when you add nodes.

You can reduce this contention if you understand how the data is accessed. Only by knowing your application can you most effectively optimize the parallel server system.

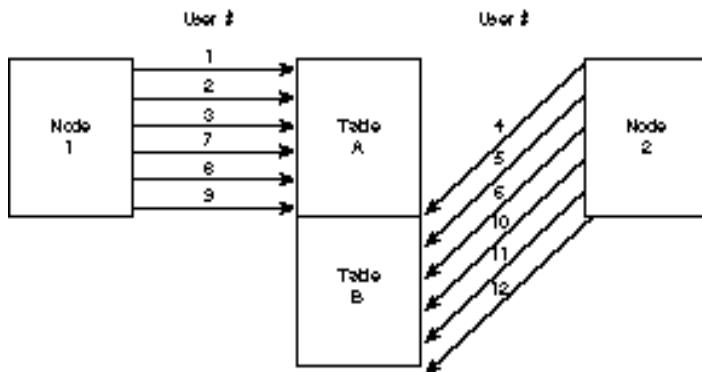
If you have a system in which some users access only certain tables and other users primarily access other tables, you can benefit from partitioning those users. If the users are spread out in a random fashion, you see a lot of pinging because multiple nodes want to update data in the same table (see Figure 21.2).

Figure 21.2
PCM lock contention.



By segmenting the users so that the users on each node access data in the same tables, you can greatly reduce PCM lock contention because each instance can hold the locks longer. Although you still have some contention, any reduction can be very useful. This kind of segmentation is necessary to get optimum performance from the Oracle parallel server system. An example of a segmented system is shown in Figure 21.3.

Figure 21.3
Reducing contention by segmenting users.



If all users have to access all the data, you can still segment this activity by using a Transaction Monitor (TM) such as the Tuxedo ETP system. A TM is class of product, known as *middleware*, used to facilitate the communication between the client and server (see Chapter 36, “Using Middleware Products”).

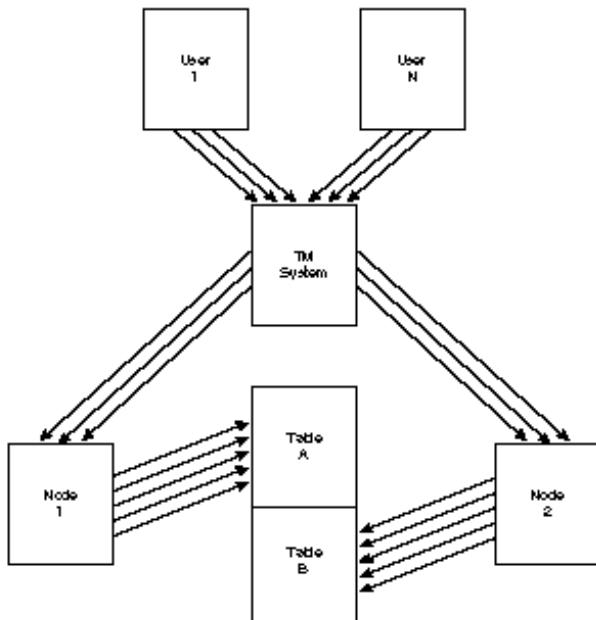
You can use the Tuxedo system on many different modes for different functions. Tuxedo has some of the following capabilities:

Feature	Description
Multiplexing	You can use Tuxedo as a multiplexer to allow multiple client processes to be serviced by a few Oracle connections, thus reducing the connections into the RDBMS.
Queuing	You can use Tuxedo as a queuing mechanism to store requests to be serviced either sequentially or at a later time.
Distributed services	Two distributed applications can be supported with the TM to transparently provide distributed services to the application.
Smart routing	The TM can be used to route requests to different servers based on the content of the request. In this way, requests can be segmented.

You use the segmentation feature to segment the data among the different servers in a parallel server system. When you use a TM, the segmentation is transparent to the users (see Figure 21.4).

Figure 21.4

Segmenting data using a TM.



Whether or not you use a TM, the main goal of the parallel server system designer is to try to set up the system to reduce lock contention by segmenting tables and users. It is a secondary goal to balance the number of PCM locks based on the amount of contention that remains.

System Design

By knowing your data access patterns, you can configure your system based on some general rules:

- ◆ **Partition work between servers.** Try to balance the systems so that users accessing the same table reside on the same computer. Doing so reduces lock contention between machines. By segmenting the work, you reduce the amount of lock traffic. Remember: Once a lock is acquired, it is released only when another system needs to lock that data.
- ◆ **Put lots of PCM locks on tables with heavy update traffic.** If you have lots of updates, you can benefit from lowering the blocks-per-lock ratio. By increasing the number of locks, you increase overhead, but by having fewer blocks per lock, you may cut down on the percentage of locks with contention.
- ◆ **Use PCTFREE and PCTUSED to cause fewer rows per block on high-contention tables.** By doing this—and decreasing the number of blocks per lock—you reduce the lock contention. However, the cost is more locks and more space required.
- ◆ **Put fewer locks on read tables.** If you have tables that are mostly read, use fewer PCM locks. Read locks are not exclusive; the reduction in locks cuts down on inter-connect traffic.
- ◆ **Take advantage of FREELIST GROUPS.** The use of the FREELIST GROUPS storage parameter allows different instances to maintain separate free lists. Use this parameter to cause inserts to occur on different parts of the data files, thus reducing contention.
- ◆ **Use read-only tables.** If a table will not be updated, make it read-only. A read-only table does not require PCM locks because no updates can be made to that table.
- ◆ **Partition indexes to separate tablespaces.** Because indexes are mostly read, you can benefit from needing fewer PCM locks. By segmenting the tables, you can put fewer PCM locks on the index tables and more on the data tables.

By properly designing the system to avoid data block contention, you reduce the number of locks that must be acquired and released, thus reducing overhead and delays.

Tuning the Parallel Server System

Not only is it important to properly design and configure the system, you must properly configure the number of PCM locks. Several Oracle parameters are used for the parallel server system in addition to the traditional Oracle parameters. Table 21.1 reviews the parameters of primary concern to performance.

Table 21.1 Parallel Server System Parameters

<i>Parameter</i>	<i>Description</i>
CACHE_SIZE_THRESHOLD	Specifies the maximum size of a cached partition table split among the caches of multiple instances. If the partition is larger than this value, the table is not split among the caches.
GC_DB_LOCKS	Specifies the number of PCM locks allocated. The value of GC_DB_LOCKS should be at least 1 greater than the sum of the locks specified with the parameter GC_FILES_TO_LOCKS.
GC_FILES_TO_LOCKS	This is a list of filenames, with a specification of how many PCM locks should be allocated to that list of files. Optionally, the number of blocks covered by a PCM lock can be specified.
GC_LCK_PROCS	Specifies the number of lock processes (LCK0 through LCK9) to create for the instance. Usually, the default value of 1 is sufficient unless an unusually high number of locks are occurring.
GC_ROLLBACK_LOCKS	The number of distributed locks available for each rollback segment. The default value is usually sufficient.
PARALLEL_DEFAULT_MAX_INSTANCES	Specifies the default number of instances to split a table among for parallel query processing. This value is used if the INSTANCES keyword is specified in the table/cluster definition.

Of these parameters, probably the most important is GC_FILES_TO_LOCKS. You use this parameter to specify, on a data-file level, how many locks to allocate. The following equation shows how to calculate the average number of locks per block:

Locks per Block = Number of Locks Allocated to the Data File / Number of Blocks in the Data File

When you use GC_FILES_TO_LOCKS in conjunction with the PCTFREE and PCTUSED creation parameters, you have some control over how many rows are affected by a PCM lock. By limiting a PCM lock to a small number of rows in a particularly active table, you can avoid locking data you don't really want to lock. By the same token, if you have very little contention, you can reduce the number of PCM locks by allocating a large number of blocks per lock.

The dynamic performance tables V\$BH, V\$CACHE, and V\$PING contain information about the frequency of PCM lock contention. Look at the FREQUENCY column in these tables to get an idea of the number of times lock conversions take place because of contention between instances.

The dynamic performance table V\$LOCK_ACTIVITY gives information on all types of PCM lock conversions. Use this information to see whether a particular instance is seeing a dramatic change in lock activity. If so, this may indicate that you don't have a sufficient number of PCM locks on that instance. With this information, you can use the V\$BH, V\$CACHE, and V\$PING tables to identify the problem area.

Summary

This chapter described how the Oracle parallel server system operates. Using that information, you determined several ways to optimize the system, including the following:

- ◆ **Partition your data.** By partitioning the system so that each server primarily accesses a certain part of the data, you minimize the amount of contention and lock usage.
- ◆ **Use a TM if necessary.** Use a Transaction Monitor to aid in the segmentation of traffic between nodes.
- ◆ **Tune PCM locks.** By effectively using PCM locks, you can reduce lock contention. Use a sufficient number of locks on heavily accessed tables and reduce the number of locks on lightly used tables.
- ◆ **Limit the number of rows per lock on active tables.** Doing so causes more locking to occur but avoids locking rows unrelated to the data you are updating.
- ◆ **Use FREELIST GROUPS.** The FREELIST GROUPS storage parameter allows different instances to maintain separate free lists so that inserts occur on different parts of the data files, thus reducing contention.
- ◆ **Use read-only tablespaces.** If possible, take advantage of read-only tablespaces to reduce the need for PCM locks.

By understanding how the parallel server system operates, you can better tune and configure the system. You can effectively design a parallel system that provides both fault tolerance and excellent performance.

Chapter **22**

Optimal Backup and Recovery

Some of you work with mission-critical systems that must be available 24 hours a day, 7 days a week. With these types of systems, the time it takes to do hot and cold backups must be minimized. By minimizing the time it takes to do a hot backup, the performance effect on the system is minimized. By shortening the time it takes to do a cold backup, system downtime can be minimized.

Backup and Recovery Terminology

Following are definitions of some of the terms used in discussions of backup and recovery.

Hot backup	Officially known as an <i>online tablespace backup</i> . This backup occurs while users are connected to the database and executing transactions (even to the table being backed up). Hot backups are performed on a tablespace basis.
Cold backup	Officially known as an <i>offline backup</i> . This backup is done when the Oracle instance is not running. Cold backups can be performed on a tablespace basis.
Full backup	A backup performed on every tablespace in the database. Typically, full backups are done as cold backups.
Export	The Oracle export facility can be used as a method of performing backups but is usually not the preferred option because of performance issues.

An Oracle backup can consist of either a hot backup (done while users are connected and running transactions) or a cold backup (the database is shut down). Neither relies on Oracle to perform any functions except to keep the data files from changing during the backup. Using export can be an alternative to traditional methods because it includes the additional feature of allowing the structure of the database to change. However, an export does not provide for the flexibility and recoverability that traditional backups do. Rather than just copying the data as it exists, you can export and import the data to restructure the database and reduce fragmentation.

Although an export can be loosely considered a backup function, the export facility does not provide the same level of recovery as the standard Oracle backup facilities. However, there is more to export/import than just backup and recovery. Export/import can be useful in moving from one version of Oracle to another and for reconfiguration of the database. Export/import can also be used to move data between Oracle databases. Because exports do not allow for instance recovery and because many of the concepts of the export process are similar to those of the traditional backup procedure, this chapter does not address the export process as a backup method.

This chapter consists of a number of hints for how to minimize the effect of the backup and increase its performance. Some of the hints may be directly applicable to you; others may not apply to your configuration. By determining which hints might help you, you can find some useful optimizations for your configuration. The chapter starts by describing some of the characteristics of an Oracle backup and from those characteristics derives methods of improving backup and recovery performance.

RDBMS Operational Review

When you make any changes to the Oracle database, the changed information is not only eventually written to the database by the DBWR process, it is recorded in the redo logs by the LGWR process. When the log file is filled, a log switch occurs, which triggers a database checkpoint.

The checkpoint event ensures that all dirty buffers in the SGA are written out to the data files on a regular basis. The checkpoint can occur more frequently than a log switch for data integrity purposes.

When the log switch occurs, the previous log file is copied to an archive log file (assuming that you are running in ARCHIVELOG mode). This archive log file, along with the online redo log file and the last backup, allows for database recovery.

Backup Process

When you perform a backup of the data files, you not only copy the data itself, you copy valuable header information that identifies the data file. The header information tells Oracle when the data file was last backed up and the point in its operation at which it was when this backup occurred.

Oracle uses the information from the backup, the archive log files, and the online redo logs to recover the data to where it was when the system failure occurred. The backup feature is one of the things that makes Oracle such a powerful and robust RDBMS system.

As part of your ongoing backup process, it is important to keep all the archive log files safe and available in case a recovery is necessary. The archive log files allow you to restore your system to the point of failure should it become necessary to recover the system.

Recovery Process

If you are not running in ARCHIVELOG mode and a failure occurs, you can reload your database from the last full cold backup and start running from there. There is no facility in Oracle to recover any further if you are not running in ARCHIVELOG mode.

If you *are* running in ARCHIVELOG mode and a failure occurs on a data file, you only have to restore the damaged data file from a recent hot or cold backup. It is not necessary to restore the entire tablespace unless you are not running in ARCHIVELOG mode.

When you bring the tablespace back online, Oracle prompts you to apply various archive log files. Using these files, Oracle restores all the changes made to this data file until the time of the media failure. In this manner, no data is lost and minimal downtime is incurred.

If you have the Oracle Parallel Query option installed and have enabled parallel recovery, the time it takes to perform instance recovery can be greatly reduced. I find the Oracle backup and recovery process to be one of the things that separates Oracle from its competition.

The Oracle redo log architecture and the archiving processes allow Oracle to recover from almost any failure. These features provide a high level of data integrity and robustness.

Characteristics of the Oracle Backup Process

Before looking at the specifics of the hot and cold backups, you should review a little of the operation of Oracle and how the backup process fits into this operation.

To characterize an Oracle backup, you should first understand how it operates in terms of both a hot backup and a cold backup. Because the cold backup is much simpler, we'll look at it first.

Cold (Offline) Backup

In a cold backup, the system is in a quiescent stage, the database is shut down, and no Oracle processes are running. The cold Oracle backup is sometimes referred to as an *offline backup*. To perform a backup, each of the data files is backed up as an OS file or raw partition. An offline backup can be accomplished using an OS function or through a third-party utility you may have purchased. It is important that you back up not only the data files but the redo log and control files as well.

The characteristic of this kind of backup is that each data file is read sequentially. Unless some other OS function accesses the disks, you get purely sequential I/O. To take advantage of the sequential nature of the cold database backup, you should isolate the sequential I/Os by avoiding any other I/Os to that disk volume. Because several tablespaces may use the same volume, you should serialize the backup of those data files.

Hot (Online) Backup

When performing a hot Oracle backup, also referred to as an *online backup*, the tablespace to be backed up is taken offline using the Oracle command `ALTER TABLESPACE BEGIN BACKUP`. The tablespace being backed up can still be modified, but all changes are kept in the SGA and are not written to the data files until the backup is complete. Once the tablespace is brought offline with this command, the data files associated with that tablespace are backed up as OS files or devices. The backup can be accomplished using an OS function or using a third-party utility you may have purchased. The control files are backed up in a slightly different manner.

To back up the control file, you use the Oracle command `ALTER DATABASE BACKUP CONTROLFILE TO 'filename' REUSE`. This command copies the current control file to the file specified by `filename`. It is important that the control file be backed up any time the layout of the database changes.

The characteristic of this kind of backup is that each data file is read sequentially. Unless some other OS function accesses the disks, you get purely sequential I/O. To take advantage of the

sequential nature of the database backup, you should isolate the sequential I/Os by avoiding any other I/Os to that disk volume. Compared to a cold backup, it is much more difficult to guarantee that no other I/Os are being done to the disk volumes during a hot backup because the database is still up and running. Because several tablespaces may use the same volume, you should serialize the backup of those data files.

Data Access Patterns During Backup

With both the hot and cold backups, there is purely sequential access to the data files by the backup process. By isolating the sequential disk access, much higher I/O rates can be accomplished than when accessing a disk volume in a random fashion. Because you want to do backups as quickly as possible, the highest I/O throughput is desired. Therefore, you should try to get as much sequential access as possible.

During a cold backup, you can control how the data is accessed on the disks by carefully designing the backup process. By serializing the backups so that there is one sequential data access per disk volume at a time, you can achieve very good backup speeds. The preceding statement assumes that the media to which you backing up can handle the load.

During a hot backup, you don't have as much control over how the data is accessed except through the design of the system. Because other tablespaces are active during the backup, if any of them have data files on the same disk volume as the one you are backing up, accesses to that disk volume will be of a random nature. If you can configure your system so that there is only one tablespace per disk volume, you can maintain the sequential data patterns of the backup process.

System Load During Backup

During the backup process, the load on the system is caused primarily by I/Os—unless you are using compression for the backup. If you are compressing the data, there is significant CPU usage. In fact, if you do a hot backup and you do not have sufficiently fast processors, you may be CPU bound during the backup and the backup may take even longer to perform.

Because the time it takes to perform the backup depends primarily on how fast you can read the data from the data files, any external factors such as compression speed or network throughput should be minimized if you are to run the backup at its full potential. Later in this chapter, you learn about some techniques you can use to overcome some of these difficulties.

Backup Goals

The goal of the backup system is to back up the data files as quickly as possible. For a hot backup, it is important to get the data files back online as soon as possible. The backup operation should be an I/O-bound task. If the performance of the system is limited by the speed of your CPUs or by the throughput of your network, take steps to minimize these bottlenecks. Although speed

is important, it is equally important for the backup to run completely without errors. Do not sacrifice completeness for speed. A fast backup that is missing files or has data corruption is useless.

By minimizing factors such as compression speed and network throughput, you can maximize the performance of the backup. The highest backup speed possible is the speed at which the data files are read in a purely sequential fashion. Of course, system overhead prevents you from reaching the theoretical maximum, but by minimizing this overhead and maximizing throughput, you should be able to approach this speed.

Both the hot and cold backup operations involve reading the data files and writing the data to the backup medium. The actual copying of the data itself is done using OS or third-party utilities. Oracle allows you to back up the system in the most optimal manner for your system.

The data is accessed in a purely sequential manner. If you isolate the I/Os either by serializing the data file backups (for cold backups) or by carefully partitioning data (for hot backups), you can achieve maximum performance. The following section looks at some of the system designs that can help you achieve maximum backup performance.

System Design Considerations

The following sections distinguish between the hot, or online, backup and the cold, or offline, backup and provide several hints on how to design your system for optimal backup performance. Some of these hints may or may not apply to your configuration. It is up to you to determine which apply to your system.

Cold Database Backup

In a cold database backup, the database is shut down and no users have access to the system. This situation is the best case for backup because it is up to the database administrator to determine how the database is backed up and the data access patterns. This control makes it easier to optimize the system than is possible with a hot, or online, backup.

Because you have control over how the disks are accessed in a cold backup, you should design the backup procedure to take maximum advantage of sequential disk I/O to the backup storage device. By making the I/Os sequential, you can achieve a much higher throughput rate. The following sections examine how you can achieve sequential I/O.

Physical Data Access

The way to guarantee sequential I/O is simply to allow only one backup stream at a time to access a set of disks. If you use Oracle disk striping or RAID arrays, the concept is the same. For each disk drive or volume of disks, serialize the backup of the data files so that only one backup process at a time accesses it. Because multiple RAID volumes may exist on your ma-

chine, you should parallelize the backup operation on these devices by backing up all the disk volumes simultaneously.

For a cold backup, it is not necessary to change the design of the system because you eliminate all other database accesses to the data files by shutting down Oracle. By designing the backup procedure, you should be able to achieve optimal I/O rates.

Hardware/Software Considerations

When choosing hardware and software for backup, make your decisions with the following issues in mind:

- ◆ **Fast tape device.** The speed of the backup is most likely limited by how fast you can write to tape.
- ◆ **Fast network device.** If you are backing up the system through a network, make sure that your network hardware is not a limiting factor.
- ◆ **Optimal software.** Some third-party backup vendors are much faster than others. Make sure that you are getting good software for your platform.
- ◆ **Optimal tuning.** By adjusting values such as the block size you are using, you may be able to optimize disk and tape speeds.
- ◆ **Compression.** If you use compression, make sure that you are using an optimal compression method that has both high performance and a good compression ratio.

By optimizing all these areas, you should be able to achieve an excellent backup rate. Shop around for tape devices; you will find that some of the new tape devices have incredible performance characteristics.

Hot Database Backup

The hot database backup is more complex than the cold database backup. In the cold database backup, there is usually no other disk activity besides the backup itself; in the hot database backup, there is significant disk activity to the other data files in the system. To optimize the backup, there are several things that can be done, all of which happen at the database and hardware design stage.

The optimizations you can make all have the goal of shortening the amount of time data files must be taken offline for the backup. These optimizations can include the following:

- ◆ **Isolate sequential I/O.** This may or may not be possible or effective. You should not put only one tablespace on a disk volume unless the performance is sufficient during normal operation.
- ◆ **Use temporary backup volumes to compensate for slower components such as network or tape.** If possible, back up your database to a local disk. After you bring your users back online, you can begin the task of backing up those files to the network or to tape.

- ◆ **Use high-speed, isolated links between the database server and the backup server.** By backing up the database using an isolated link, you can improve the performance of the backup as well as limit the effect the backup has on the network bandwidth.
- ◆ **Split up backups.** It is not necessary to perform the entire backup at one time. If necessary, break the backup across several days.

These optimizations can help speed the backup process. Not all these hints may apply to your configuration; use the ones you think will benefit your particular system.

Physical Data Access

The following sections describe a few of the ways you can better configure your system to optimize the backup process. Some of the methods described may be impractical for your installation because of the additional hardware requirements. But by understanding the concepts, you may be able to improvise a method that better suits your particular installation.

Isolating Tablespaces

By isolating data to a single tablespace per disk volume, you can take advantage of the sequential nature of the backup's I/O access patterns (assuming that very little read activity is occurring). Although this may seem to be a good way to get high-speed sequential I/Os from the disks, the system may suffer in the long run.

Because the tablespaces must also be used during the normal operation of the system, it may not be wise to limit the disk volumes to one tablespace. When you do this, you may have disk volumes that are either overutilized (causing an I/O bottleneck) or underutilized (causing some other disk volume to be overutilized).

For many applications, the best way to balance I/Os is to spread the tablespaces over all the available disk volumes. By having all the tablespaces on all the disk volumes, you have the best chance of balancing the I/Os, even if some tablespaces are more active than others.

Isolating the tablespaces so that you have only one tablespace per disk volume may help backup performance; however, the overall performance of the system may suffer. I cannot recommend that you have only one tablespace per disk volume unless backup time is more critical than general RDBMS performance.

NOTE: Even if you have only one tablespace per disk volume, because the backup only defers to the SGA in cache reads and writes, there is no guarantee that the access to the disks will be sequential if there is read activity on the tables within the tablespace that is being backed up.

Use of Temporary Backup Space

Because access to the tablespace is deferred to the SGA for updates during the backup, it is essential that you finish the backup as soon as possible to get the table back to its normal state.

If you use a tape drive for the backup media, you may be constrained by the speed of the tape drive. If you use a network to back up the data to a network backup server, you may be constrained by the speed of the network. In both of these cases, you can minimize the time that the tablespace access must be deferred by backing up to a disk drive or set of disk drives reserved for this purpose. Even though you may not be able to get purely sequential I/Os out of the data files, you will be able to take advantage of sequential access to this temporary space.

Once the data has been written to this temporary space, the tablespace can be brought back online and normal operations resumed. At this point, you can back up the data from the temporary space to the tape or network—or even compress the data—without significantly affecting performance to that tablespace.

Once the data has been transferred off the RDBMS system, you can back up the next tablespace. In this way, you minimize the time that database operations are affected by the backup. Of course, some extra I/Os are associated with the backup files, but these are isolated to a separate disk volume. Compressing the data while it is still on the RDBMS system can significantly affect performance by taking large amounts of CPU time.

Isolate the Network

When performing backups across the network to another server or a dedicated backup server, it is important to keep in mind the large amount of network bandwidth that can be consumed by this backup. Because large amounts of data are being transferred with large block sizes, the load can be significant.

By isolating the backup to its own network segment, you can not only increase the performance of the backup itself but also reduce the effect your backup has on other users on the network. Confining the backup to its own network segment isolates the effect of the backup to the segment used for the backup.

If multiple servers use the same network segment for backups, simple scheduling can reduce the load by spreading out the backups so that each system backs up at different times. If you need very fast backups to multiple systems, you may have to segment the network further.

Hardware/Software Considerations

When selecting hardware and software for your backup tasks, try to get the best-performing hardware currently available. Because backup is such a critical task, it is worth the extra expense to get the very latest hardware and software.

If you can perform only hot, or online, backups, you may want to invest in some extra disks to off-load the data to temporary storage so that you can get the data files back online as soon as possible. If you perform your backups over a network, you should get fast network hardware such as 100Base-T or fiber-optics network hardware.

Your backup media (such as tape or optical storage) should perform at rate necessary to do the backup in the specified amount of time. Upgrading your backup hardware to a faster tape device is always a good investment because both the speed and quality of your backup is essential to your system.

TIP: Be sure to periodically test your backups to make sure that they function properly. The time of a failure is the wrong time to realize that your backups have been failing.

Review of Design Considerations

By carefully designing a backup plan that isolates sequential I/Os on a per-disk-volume basis, you can greatly enhance the performance of the cold backup. With hot backups, it is more difficult to isolate the I/Os because read activity still occurs on the data files during backup and because other data files may share the same disk volume.

By taking advantage of fast backup media (such as fast tape devices, optical storage, and fast network components), you can enhance the performance of the hot backup as well. If you are innovative and use a temporary disk storage device, you can reduce the time the tablespace is offline and bring the system back to its full performance level as soon as possible.

Tuning Considerations

When tuning for backups, there are no particular Oracle parameters that can help you because Oracle allows you to use your OS or third-party backup utility to perform the backup itself. But there are a few things you can do to improve the performance of the backup program itself.

Test various block sizes. Don't stop with the default block size that the backup program uses. By increasing the block size used both to read the data from disk and to write the data to tape, you may find enhanced performance. Increasing the value too high may degrade performance.

The optimal block size for backups varies based on your operating system, disk hardware, tape hardware, and OS. A good starting point is a 64K block size. By varying the block size and timing the backups, you should be able to determine the kind of effect different block sizes have on your system.

If the time it takes to perform a backup is critical, and you are not particularly concerned about the overall effect on the system, you can increase the priority of the backup process. Doing so ensures that the process always runs when it is ready; in some OSes, increasing the backup's priority ensures that the job is never preempted. Although this may help reduce the time it takes to perform the backup, I do not recommend adjusting process priorities. Doing so can sometimes degrade performance.

System Enhancements To Improve Backup Performance

You can enhance the system by adding faster backup devices such as faster tape devices, faster network hardware, or faster software. By using the fastest storage medium, you can significantly reduce the time it takes to perform the backup.

Don't assume that all backup software performs the same. The way the software reads the data files—as well as the algorithms it uses for compression—can greatly affect performance. By choosing a backup software package that performs well and has all the features you need, you can optimize your backup performance.

CPU Enhancements

Enhancing the CPUs on your SMP or MPP system can provide instantaneous performance improvements (assuming that you are not I/O bound). The speed of CPUs is constantly being improved, as are new and better cache designs.

Although the backup process primarily focuses on reading the disk and writing to another device, many backup programs include efficient compression options. Because you usually are limited by the speed of your tape drive and the speed of the network for network backups, compressing the data allows you to increase the performance of the backup by reducing the amount of data that must be transferred. If you have sufficiently fast CPUs, you can perform efficient compression. If you add CPUs to your system, you can reduce the effect the backup or compression has on other users in the system.

Make sure that the speed of the CPUs is sufficiently fast to compress the backup data at a rate that meets or exceeds the performance of the slowest component in the backup process. Doing so guarantees that the compression process is not a bottleneck in backup process. The way to test this is simple: Back up a typical table twice: once using compression and again without using compression; then compare the results.

I/O Enhancements

By enhancing the I/O devices to be optimized, you can see significant performance increases during backup. Disk arrays allow you to perform a hot backup much more effectively than

with traditional Oracle striping techniques. The use of a disk array can be enhanced by using a sufficiently large block size to cause multiple disks to be active simultaneously.

Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk as well as to mask the overhead associated with fault tolerance. Backup activities may see improved performance with disk controllers that take advantage of read-ahead features.

Read-ahead occurs when the controller detects a sequential access, reads an entire track or some other large amount of data, and caches the additional data in anticipation of a request from the OS. Because this data can then be accessed almost immediately, having data in the controller cache can be a benefit.

Enhancements to the I/O subsystem almost always help backups because large amounts of data are read during backups. Be sure that you have a high-speed I/O path, properly configured. An I/O bottleneck is usually difficult to work around. A good third-party backup program can also be beneficial.

Network Enhancements

If you perform backups over a network, the speed of the network is extremely important, as is the amount of other traffic on the network. By segmenting the network into a production OLTP network and a backup network, you can see a reduced effect on the OLTP users and an increase in backup performance. This network segment should be of a high-performance variety, such as 100Base-T or fiber optics.

The less time you spend waiting for the network, the less time the backup takes. Because it is easy to improve network performance by adding additional links or faster links, it is a good investment to try this.

Split Up the Backup

One way you can minimize the time the system's performance is reduced because of backups is to split the backup into several portions. The backup can be done on a tablespace basis: you can back up some tablespaces on one day and back up other tablespaces on other days. Doing so can reduce the amount of time required to perform the backup on a daily basis. However, the total time necessary to perform the backup remains the same.

When segmenting backups, you can decide which tablespaces have more critical data or are updated most frequently and back up those tablespaces more frequently than other, less-used tablespaces. In this manner, you can reduce the time it takes to restore your most critical data.

In the event of a system failure, remember that the time it takes to restore your data depends on when the last backup occurred and how much data has been modified or added to that tablespace since that backup. By performing backups more frequently on active tables, you can reduce the time it takes to restore the system.

A reasonable schedule would allow you to perform cold backups during non-peak hours. You can split up the backup into several sessions. By doing this, you can perform a backup over several days, reducing the time each day that the system's performance is affected. If backup takes several days, you may want to back up the most active, critical data more often.

Here are some recommendations for segmenting backups:

- ◆ **Perform hot backups in off-peak hours.** Because the hot backup affects performance somewhat, the backup should be arranged so that it affects the fewest number of users.
- ◆ **Make sure that your data is protected.** Even if users are affected, you must maintain a level of protection that allows you to get back online as soon as possible in the event of a system failure. Wouldn't you rather have a few users grouse about the inconvenience caused by frequent backups than an entire corporation up in arms because you can't restore their data after a system failure?
- ◆ **Back up heavily used data more often.** The time it takes to recover depends on when the last backup occurred and how much data has been modified since that time.
- ◆ **Back up critical data more often.** If a system failure occurs, it may be important to get some critical tables back online first. If this is true in your implementation, you should back up those critical tables more often.

These recommendations should help you design a reasonable backup schedule that will allow you to segment your backups and at the same time achieve reasonable recovery times. Of course, your system may have its own special requirements that must be addressed.

Review of System Enhancements

By carefully examining the characteristics of the backup system, you can enhance the performance in several ways:

- ◆ Additional CPU power can increase the performance of compression utilities, reducing the amount of data that must be written to tape or network.
- ◆ I/O enhancements, such as disk array, can help performance for both hot and cold backups.
- ◆ Network enhancements, such as a high-speed link, can reduce the time it takes to transfer the data between machines.

- ◆ Splitting up backups across several days' off-peak hours can reduce the overall effect of the backup on users while maintaining a reasonable recovery time.
- ◆ Other innovative enhancements such as hardware compression devices can be used to improve backup performance.

By taking advantage of enhancements to your system, you can improve the performance of your backup and thus reduce the time the backup takes. In the case of a cold backup, this reduces the time the RDBMS is offline.

Performance Verification

Once you design and build your system, you must have some way of verifying performance. If you build your system for optimal backup/recovery performance, this task is quite simple.

The simplest method of verifying performance is to test the system using the backup process as the standard. It is not necessary to save a test database to test backup performance; simply measure the time it takes to perform your daily backups. Keep in mind that as the amount of data in your database grows, the time it takes to back up those tables also increases.

By periodically documenting the backup performance of the system and trying new parameters such as various block sizes, you may be able to fine-tune your backup process. I don't recommend that you change any parameters on a production system that might put the backup at risk, but values such as block size incur no risk. The following sections look at what should be tested in the RDBMS and the operating system.

What To Test in the RDBMS

Whether you perform hot or cold backups, there is nothing in the RDBMS itself you can tune to increase the performance of the backup. However, this is not the case if you are using the export utility to back up the system.

The export process *does* use the RDBMS to perform its function. The function of the export utility is to read data from the database and output that data to an OS file. An export does not require much tuning except in the case of long rows (in this case, you may want to increase the **BUFFER** parameter).

By increasing the size of the **BUFFER** parameter, you may see some performance improvement. Perform the same export using different values for the **BUFFER** parameter and note any differences in performance.

What To Test in the OS

You can perform various tests in the OS to verify and increase the performance of the backup process:

- ◆ **Data disk read performance.** By testing the process of reading from the data disk drives, you can determine an optimal block size. Many disk drives and disk array controllers have a certain block size that is optimal.
- ◆ **Tape write performance.** By writing data to the tape drive using various methods and block sizes, you can determine the optimal configuration.
- ◆ **Network configuration.** It may be possible to tune the network so that larger network packets can be used to cut down on the number of packets that must be sent.
- ◆ **Optimized backup software.** By using the most efficient OS facility or third-party utility, you can increase performance of the backup.
- ◆ **Optimized compression techniques.** Using a modern, high-performance compression algorithm can improve performance and efficiency.

An improvement in any of these areas can add an incremental performance improvement to your backup process. The following sections examine some of these areas in more detail.

Disk Block Size

Many disk drives and disk array controllers have a certain block size that is optimal. By reading the data file and discarding the data, you may be able to find the optimal block size for your particular configuration. With some disk array controllers, having a sufficiently large disk request allows multiple disks to service that request simultaneously.

During a backup operation, very large block sizes (such as 64K) are not unreasonable. With some devices, a block size even larger than this might be acceptable. When the block size gets too large, performance begins to degrade because internal OS buffers have been overflowed or the request is split up by the controller's device driver into smaller pieces, causing additional overhead. By testing various block sizes, you can determine what is optimal for your specific configuration.

Tape Block Size

Different types of tape drives have different characteristics. In determining the proper block size for your tape drive, the goal is to keep the tape streaming. When the tape drive is streaming, it is continually moving and writing data to the tape. If the tape drive is not fed data quickly enough, the tape stops, rewinds a short amount, and starts up again when more data arrives.

Even though data may be arriving at the tape device at a certain rate, the rate at which it is written to tape may be degraded because the tape is stopping and starting. This phenomenon is known a “shoe-shining” or “rocking” because the tape must move back and forth over the tape head for the data to be written contiguously to the tape.

By feeding data to the tape at a fast enough rate to stream the tape, you can see significant improvements in performance. Any time the tape has to stop and back up, performance is degraded. Providing a large enough block size and buffering data is important in helping improve tape performance.

Try various block sizes when the tape is streaming to see what is optimal for your particular hardware. If the tape is not streaming, this test is not really valid because performance is severely degraded by the shoe-shining effect.

Network Packet Size

By using larger network packets, you can transmit the same amount of data with fewer packets. Because there is an inherent overhead associated with each network packet, reducing the number of packets enhances the overall performance. The size of the network packets is typically not an OS issue but an issue of the backup software you are using. It may be necessary, however, to increase the maximum allowable network packet size to accommodate your backup software.

If you use larger network packets, you can typically use more of the bandwidth of the network. When a typical Ethernet network card wants to transmit on the network, it first checks to see whether there a packet is currently being transmitted. If the network is busy, the card waits a specified period of time and then tries to transmit again. If you use many small packets, it is more likely that the network will be busy than if you use a few large packets; with the larger packets, there is usually more time between transmissions.

The efficiency of larger packets provide better performance. If network bandwidth is still an issue, consider adding an isolated segment for backups or going to a faster network controller such as 100Base-T or fiber optics.

Backup Software

The choice of OS or third-party backup programs can be very important. Don’t be influenced solely by how flashy the GUI is or the number of features in the backup software. The features included in many of the backup packages are important but should not overshadow the product’s performance.

The main goal in backing up your database is to protect your data and to disrupt the user community as little as possible. To do this, the backup software should be high performance and reliable. The integrity of your system depends on the reliability of your backups. Measure the performance of your backup software periodically and test the backups to make sure that everything is working properly.

The performance of your backup software can depend on the configurability of the product. Make sure that you can adjust the block size of both data reads and tape writes so that you can optimally tune for your particular devices.

There have been vast improvements in compression algorithms over the last few years; some of the new techniques can be quite effective in improving both performance and efficiency, as described in the following section.

High-Performance Compression

Compression techniques have improved significantly in the last few years. This improvement affects both the performance and efficiency of the compression.

By using a high-performance compression algorithm in your backup procedure, you can improve the efficiency of the backup. By using a recent version of your backup software, you will most likely take advantage of the most optimal compression techniques. Many of the components of the backup (such as the network or tape device) have a maximum data rate. By using compression to reduce the amount of data to be written to these devices, the entire data file can be written to the device faster.

Your performance may suffer if the compression causes you to become CPU bound. If you have a sufficient number of CPUs that are fast enough to keep the tape or network device running at maximum efficiency, compression can be a beneficial part of your backup procedure.

You can verify the performance of your CPUs by backing up the same data file both with and without compression and comparing the performance. Keep in mind that many tape devices have hardware compression built in. If you compress the data before sending it to the tape device, the efficiency of the hardware compression may be reduced, nullifying any beneficial effect of OS or software compression.

The only way to know whether compression can be beneficial is to try backing up with and without it. Once you have determined the beneficial value of compression, you can concentrate on other areas of enhancement.

Summary

This chapter looked at how backups work in the Oracle system and some of the ways you can enhance the performance of your backups. With mission-critical systems that must be available 24 hours a day, 7 days a week, backup and recovery performance is crucial. By increasing the performance of the backup, you shorten the amount of time that users see some sort of degraded performance caused by the backup.

Chapter 23

Miscellaneous Configurations

This chapter looks at several miscellaneous configurations and describes how you can optimize these configurations. By looking at how the application accesses the data in the system and takes advantage of Oracle features, you can determine the best way to optimize that configuration. The following chart lists the configurations described in this chapter and the applications associated with each.

<i>System Type</i>	<i>Description</i>
Financial systems	Financial applications. These systems are used to run the financial operations of your business. These systems use financial applications such as Oracle Financials or SAP.
Replicated systems	The replicated system is designed to duplicate copies of tables or subsets of tables to permit local access to this data. This chapter describes systems using Oracle's Symmetric Replication package.
Distributed systems	The distributed system allows for a set of servers to create a single view of a database, even though parts exist on different machines. This chapter describes systems using Oracle's Distributed option.
Oracle TextServer 3.0	TextServer 3.0 with ConText is designed to allow for storage and processing of text documents. This system is a server for the Oracle TextServer product.
Oracle Office Server	The Oracle Office system is designed to provide office services such as messaging, scheduling, and so on.
Oracle WebServer	This system has new Oracle features to provide World Wide Web services from an Oracle database.

This chapter provides a brief overview of each type of system, lists the characteristics of that system, and provides a brief discussion on optimal configuration.

Financial Systems

Financial systems provide a rich set of applications used extensively in businesses around the world. These applications are sometimes known collectively as *Oracle Financials*. The Oracle Financials application set is built around the Oracle RDBMS and includes applications such as Oracle Human Resources, Oracle Payroll, Oracle Payables, Oracle Receivables, Oracle General Ledger, Oracle Assets, Oracle Inventory, Oracle Order Entry, Oracle Purchasing, Oracle Bills of Material, Oracle Engineering, Oracle Master Scheduling, Oracle Work in Progress, Oracle MRP, Oracle Capacity, Oracle Project Costing, and Oracle Project Billing.

These applications run many of the world's mission-critical businesses. The applications are available for most platforms; both character-based and GUI-based versions of these applications are available.

Other financial systems that use the Oracle RDBMS include the SAP system. Although the applications themselves function quite differently from the Oracle applications (from the point of view of the RDBMS), they tend to operate in a similar fashion.

System Characteristics

In general, financial systems have some characteristics of OLTP and decision support systems and some characteristics that are unique to financials:

- ◆ **Concurrent access.** In these systems, online users perform different activities against the same database.
- ◆ **Batch jobs.** These batch jobs may involve inventory reports, payroll, manpower studies, and so on.
- ◆ **Large amounts of data.** These systems tend to be fairly large and heavily used.
- ◆ **Random data access.** The OLTP-type transactions cause large amounts of random I/O on the system.
- ◆ **Sequential data access.** The DSS-type queries cause large amounts of sequential I/O on the system.

During the day, the system may have many users simultaneously inputting data concerning accounts receivable, accounts payable, and human resources; at other times, the system is accessed more like a DSS system for reporting or payroll.

You experience both of these characteristics in this type of system. If possible, you should perform the reporting and payroll activities at off-peak times when the system is not as heavily loaded.

Design and Tuning Hints

Because the financial system tends to have characteristics of both the OLTP and DSS systems, the design of the system should take into account both characteristics. In designing this system, use the following goals to achieve optimal performance:

- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out across many drives, you can increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

All these factors contribute to the optimal data layout of the system. To achieve these goals, first look at the ways you can optimally organize the system.

Design

The main goals in designing the physical data layout are to provide balanced I/Os across all the disks that are randomly accessed and to isolate the sequential I/Os. You know that the financial system has characteristics of both the OLTP and DSS systems.

An OLTP system has heavy update activity that leads you to isolate the redo log and archive log files. You don't do this with a DSS system because the DSS doesn't have heavy redo activity (although it certainly does not hurt the performance of DSS system to have isolated log volumes).

For both OLTP-type transactions and DSS-type queries, it is important to have a sufficient number of disk drives to handle the I/O generated by these transactions. It is important to monitor the system's I/O rates to ensure that the I/O subsystem is not overloaded during peak times.

The layout for a financial system looks more like an OLTP system than a DSS system because of the log activity. A minimal configuration should look something like this:

<i>Element (# of Volumes)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance.
Redo log (1+)	You should have at least one log volume. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance is degraded during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you need to archive. This data can be written to tape as necessary.
Data and index (1+)	Because you always get concurrent access to disks with a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, "Advanced Disk I/O Concepts," remember that you can only push a disk drive to a maximum random I/O rate.

I recommend taking advantage of a disk array rather than relying on Oracle striping. A hardware disk array can provide you with the highest level of performance and optional fault tolerance, if desired.

The fact that some tables are hot and others are cold is irrelevant because the data access is uniformly distributed across all the disks. Load balancing is not an issue because the small size of the stripe ensures that random data accesses are spread out across all the disks in the array.

If you use a disk array, many of the management tasks and load balancing tasks are greatly simplified. With the disk array, you also have the option of using fault tolerance without affecting system performance. Of course, using fault tolerance requires significantly more disks.

Tuning

Tuning a financial system is much like tuning the OLTP system, with the exception of the data block size. This element has the characteristics of the DSS system. When tuning a financial system, you should analyze the system to see whether adjustments to these parameters are necessary:

- ◆ **DB_BLOCK_BUFFERS.** Buffers are probably the most critical area in the financial system. An insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck. The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The ratio of PHYSICAL READS to DB BLOCK GETS and CONSISTENT GETS is the cache-miss ratio. This number should be minimized.
- ◆ **Library cache.** Check the V\$LIBRARYCACHE table, which contains statistics on how well you are using the library cache. The important columns in this table are PINS and RELOADS. A low number of reloads relative to the number of executions indicates a high cache-hit rate. You should be able to reduce the library cache misses by increasing the amount of memory available for the library cache. Do so by increasing the Oracle tunable parameter SHARED_POOL_SIZE. Try increasing the size of the shared pool by 10 percent and monitor it again. If this is not sufficient, increase the size by another 10 percent and continue.
- ◆ **Open cursors.** You may have to increase the number of cursors available for a session by increasing the Oracle parameter OPEN_CURSORS.
- ◆ **Cursor space for time.** If you have plenty of memory, you can speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR_SPACE_FOR_TIME to TRUE.
- ◆ **Spin counts.** By tuning the Oracle initialization parameter SPIN_COUNT to enable spinning on resources (that is, the process spins instead of going to sleep while waiting for a resource to become available), you may see improved performance. SPIN_COUNT is useful when you have multiple CPUs and short-running transactions.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, look at the dynamic performance table V\$ROWCACHE. The important columns to view in this table are GETS and GETMISSES. The ratio of GETMISSES to GETS should be low.
- ◆ **Rollback contention.** Rollback contention occurs when too many transactions try to use the same rollback segment at the same time and some of them have to wait. You can tell if you are seeing contention on rollback segments by looking at the dynamic

performance table V\$WAITSTAT. Check for an excessive number of UNDO HEADERs, UNDO BLOCKs, SYSTEM UNDO HEADERs, and SYSTEM UNDO BLOCKs. Compare these to the total number of requests for data. If the number is high, you need more rollback segments.

- ◆ **Use many small rollback segments.** Create many small rollback segments; perhaps you can make a rollback segment available for each server process.
- ◆ **Create some large rollback segments.** For some of the larger transactions, you may have to create some larger rollback segments. Assign these rollback segments to the longer-running transactions.
- ◆ **Latch contention.** Latch contention can be determined by examining the dynamic performance table V\$LATCH. Look for the ratio of MISSES to GETS, the number of SLEEPS, and the ratio of IMMEDIATE_MISSES to IMMEDIATE_GETS. If the miss ratio is high, reduce the size of LOG_SMALL_ENTRY_MAX_SIZE to minimize the time any user process holds the latch; alternatively, increase the value of LOG_SIMULTANEOUS_COPIES to reduce contention by adding more redo copy latches. If neither of these solutions helps, set the initialization parameter LOG_ENTRY_PREBUILD_THRESHOLD. Any redo entry of a smaller size than this parameter must be prebuilt, which reduces the time the latch is held.
- ◆ **Checkpoints.** Under certain circumstances, you may have to tune checkpoints. Although this is usually not necessary, if you see severely degraded performance during checkpoints, you can reduce the effect by enabling the CKPT process. Do so by setting the Oracle initialization parameter CHECKPOINT_PROCESS to TRUE.
- ◆ **Archiving.** By now, you should be convinced that you should always run with archiving enabled. By adjusting the initialization parameters LOG_ARCHIVE_BUFFERS and LOG_ARCHIVE_BUFFER_SIZE, you can either slow down or speed up the performance of archiving. By speeding up archiving, the effect on the system is of a shorter duration but is more noticeable. Slowing down archiving lengthens the duration but reduces the effect. This process is described in more detail in Chapter 9, “Oracle Instance Tuning.”

As you can see, most of these recommendations are very similar to those given for the OLTP system (refer to Chapter 16, “OLTP System”). With the financial system, you can make some of the same performance enhancements as with the OLTP system; the following section reviews the possibilities.

Enhancements

You can make some additional enhancements to improve the performance of the system used for financial applications:

- ◆ **Block size.** Unlike an OLTP system, the financial system can benefit from a larger block size because of the reporting activity and the larger size of the system. I recommend trying a block size of 4,096 or 8,192.

- ◆ **Clusters.** Depending on the application, you may or may not benefit from clusters. The benefit you receive depends purely on the data access patterns and your application.
- ◆ **Hash clusters.** Financial systems typically use tables that are accessed by some sort of ID, such as an account ID. If this is the case in your financial application, you can enhance performance by creating hash clusters on some of those tables. However, if this same data is updated continually or additions are frequently made, hash clusters may result in some performance loss.
- ◆ **Indexes.** Depending on how the data is accessed, you may see significant performance benefits from indexes.
- ◆ **Multiblock reads and writes.** Because financial systems do some reporting and large queries, multiblock reads can be beneficial. The size of the I/Os used in multiblock reads should be 64K.
- ◆ **Parallel Query option.** The Oracle Parallel Query option may offer some benefits for reports or queries of long duration. I recommend using the Parallel Query option.
- ◆ **Parallel Server option.** In this type of environment, you may be able to take advantage of the Parallel Server option. How well it performs depends on how your system is designed and how the users are segmented.

Many of these enhancements can help your performance. The specific effect on your system will vary, but this information works well in most cases.

Review of the Financial System

As you have seen, the financial system has characteristics of both an OLTP and a DSS system. By taking both of these characteristics into account and looking at the data access patterns, you can design the system to work optimally for both environments. Of course, you must make some compromises to accommodate both system types, but these may be minor.

The financial system must be concerned with the user response times as well as the load of large batch mode jobs. In most cases, the system should be built to favor the online users. The larger jobs can usually be deferred until off-peak hours or run at a lower priority to accommodate other users in the system.

These systems are used for mission-critical activities, running the financial activities of many of the world's largest companies. As a robust and secure RDBMS system, Oracle is well suited for this task.

Replicated Systems

With the introduction of Oracle version 7.0, the RDBMS has facilities for replicating data among servers. There are three ways Oracle provides for replication:

- ◆ **Read-only snapshots.** Data is duplicated on a read-only basis. The data is taken periodically from a master database and copied as a snapshot.
- ◆ **Updatable snapshot.** An updatable snapshot is also a snapshot of master database but may be updated.
- ◆ **Symmetric replication facility.** This facility allows multiple copies of a database to be independently updated and kept in sync. (The symmetric replication facility has been available since Oracle version 7.2.)

These replication techniques have many common characteristics and some unique ones. By studying the characteristics, you can derive some useful tuning tips.

Replicated systems can most effectively be used to provide local access to data, eliminating network traffic. By providing the data locally, you can not only eliminate the time spent waiting for data from a remote system, you can also continue to access the data even when the link has been temporarily disabled.

The type of replication you use depends on your particular needs. If you access the data in a read-only fashion, it is much more efficient to use read-only snapshots. If the data must be kept in sync, you may want to use the symmetric replication facility. The amount of overhead associated with replication increases with the amount of update activity. It is up to you to determine what kind of replication, if any, is right for your installation.

You can also use replication to provide a backup server mechanism. By replicating a system to another server, the replicated server can be used as a backup in case the primary server fails totally.

System Characteristics

The characteristics of a replicated system vary. In addition to considering the types of transactions being performed on the system, you must also consider the replication overhead. The replication overhead has these characteristics:

- ◆ **Moderate to heavy network usage.** Depending on the frequency and amount of replicated data that must be transmitted, the network activity may be significant.
- ◆ **Update activity.** The replicated data is always in the form of an update. It is not necessary to transmit data that has not changed.
- ◆ **Batch jobs.** The replication updates are processed as batch jobs. The amount and frequency depends on the system load.

Replication adds some overhead to a system. If you use or intend to use replication, you should design some additional CPU and I/O capacity into the system.

When designing a replicated system, take care to ensure that sufficient capacity is available in the network. If necessary, add network segments to provide more capacity for the replication facility.

Design and Tuning Hints

The replicated system tends to have the characteristics of the type of application it is running. It is rarely necessary to make major design changes solely for the replication features. However, you should keep the standard design goals in mind and make sure that networking and CPU usage is considered. In designing this system, use the following goals to achieve optimal performance:

- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out across many drives, you increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.
- ◆ **Avoid saturating the network.** Because replication data is transmitted over the network, make sure that you have sufficient bandwidth.
- ◆ **Have sufficient CPU and I/O capacity.** Because the replication functions add overhead to the system, make sure that you have enough spare CPU horsepower and I/O bandwidth to accommodate this addition.

The optimal layout of the system is based on the application type. If you are performing OLTP functions, use a design that provides optimal OLTP performance. If you are performing DSS functions, optimize for DSS. Keep in mind that even with DSS-type applications, you will have significant update activity from the replication itself. These factors all contribute to the optimal data layout of your system.

Design

The main goals in designing the physical data layout are to provide balanced I/Os across all the disks that are randomly accessed and to isolate the sequential I/Os. Base the design of your system on the type of application you are running and factor in the additional update activity of the replication function. In a DSS-type system, remember to isolate the redo log files and archive log files because there is significant update activity from the replication functions.

A minimal configuration should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance.
Redo log (1+)	You should have at least one log volume. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance is degraded during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you need to archive. This data can be written to tape as necessary.
Data and index (1+)	Because you always get concurrent access to disks with a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates.

Although these recommendations are based on the use of a disk array, traditional Oracle striping across individual disks also works. Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks.

As you can see, the design of a replicated system is similar to those of other systems. The only difference in the design is providing for the additional network, CPU, and I/O load generated by the replication function.

Tuning

The type of tuning needed for a replicated system is determined by the type of applications to be run on the system. Rather than re-hash all the tuning parameters you should consider, the following list touches on parameters of interest to the replication function itself.

For the replication function, monitor the following parameters to verify that you are not running out of resources:

- ◆ **DB_BLOCK_BUFFERS.** Because there is additional update activity caused by the replication function, make sure that you have a sufficient number of block buffers.
- ◆ **Library cache.** Check the V\$LIBRARYCACHE table, which contains statistics on how well you are using the library cache. If necessary, increase the value of SHARED_POOL_SIZE.
- ◆ **Data dictionary cache.** Make sure that the data dictionary cache is of a sufficient size.

- ◆ **Rollback contention.** Because the replication updates are done in batch jobs, you may have to add some large rollback segments or increase the number of rollback segments in the system.
- ◆ **Latch contention.** Monitor the latch contention in your system and take action if latch contention is an issue. Chapter 9, “Oracle Instance Tuning,” describes how to determine whether there is a problem and how to correct it.

Use this checklist in addition to whatever tuning checklist you have compiled for the main function of your system. This list is concerned only with the replication facility.

Review of the Replicated System

The replicated system is configured and tuned based on the type of application it is running. The exceptions to this are in the following areas:

- ◆ **CPU usage.** The system may require additional CPU capacity to handle the overhead of the replication functions.
- ◆ **I/O usage.** The system may need more disks to provide for the additional I/Os caused by the replication function.
- ◆ **Network capacity.** Because the replication function occurs over a network, it may be necessary to increase the capacity of the network to compensate for the increased load.

Except for the additional overhead of the replication, treat the system as you would any other RDBMS system.

The use of Oracle replication services may or may not be a benefit to you, depending on your application and data access patterns. With Oracle replication, you may be able to think up new and innovative ways to run your business that take advantage of these features. By using replication, you may be able to distribute your business data more effectively. Which replication option you choose and how you implement it depends on your business. If at all possible, take advantage of read-only tables.

Distributed Systems

In a distributed server system, two or more systems work in tandem to appear as one server. One large database may in fact be distributed across several servers. Each server in the system is maintained independently; together, they form a consistent view of the data.

Distributed systems frequently take advantage of two-phase commits to guarantee that transactions are atomic. Without the two-phase commit, it would be very difficult to ensure data consistency and integrity.

Depending on the configuration of your system, you may have some local transactions, some remote transactions, and some distributed transactions. These types of transactions are defined as follows:

- ◆ **Local transactions.** Those transactions that occur only on the local node.
- ◆ **Remote transactions.** Those transactions that occur only on a remote node.
- ◆ **Distributed transactions.** Those transactions that affect data on both the local and remote nodes.

All three of these transaction types are available in a distributed database. It is desirable to make this fact transparent to the end user. The end user does not care whether the transaction was remote, performed locally, or of a distributed nature.

A distributed system sometimes takes advantage of data replication (refer to the preceding section). If you combine a distributed system and replicated system, not only is the system load distributed, it is protected by the replication as well.

System Characteristics

The characteristics of a distributed system are based entirely on the characteristics of the type of transactions being run on it. The distributed system has no unique characteristics except for slightly higher requirements for a high-performance, reliable network interface.

Because much of the work done relies on other servers (for example, two-phase commits), it is important to have a high-speed link between the servers. This requirement is the only major differentiating factor for the distributed system.

Design and Tuning Hints

A distributed system is configured and tuned entirely based on the application running on that server. You should make no special tuning or design changes for the distributed system, except in the networking area.

Because the distributed transaction may involve either the client machine or the server in executing a two-phase commit, you should optimize the performance of the network both from a client-to-server and server-to-server perspective. If necessary, you may want to add a dedicated link between servers and provide a high-speed network between clients and servers.

Whether or not the network must be enhanced depends on your configuration and load. In any case, enhancing the network can be easy: simply add faster components or segment the network.

Review of the Distributed System

There is very little that is necessary to enhance the performance of a distributed system. Base your tuning and configuration efforts on the type of transactions that will be run on this server.

Distributed systems are common and can be useful for providing load-sharing among servers and access to a wealth of nonlocal data. The main task of the distributed system is to appear to the user as a single large database. The performance of the overall system is enhanced by allowing multiple systems to service requests.

By splitting the work between multiple servers, each system can focus on a subset of the data. Depending on the configuration of your system, you may have some local transactions, some remote transactions, and some distributed transactions. The fact that the database is distributed is transparent to the user.

TextServer 3.0 Systems

The Oracle TextServer 3.0 system is an Oracle add-on designed to assist with the storage, retrieval, and processing of textual data. TextServer 3.0 works with text data stored in an Oracle database and text external to the database. This chapter describes only the text data stored in the Oracle database itself.

Using TextServer 3.0, you can not only store and retrieve text but perform additional useful functions such as indexing and textual search functions. Searches of the textual data can be done with a word search using a pattern-matching algorithm; alternatively, you can use a theme search if you need a thesaurus to match a meaning.

TextServer can be used by itself or with Oracle ConText, a text-processing system that allows for searches based on context, phrases, or themes. Because it is not feasible to search the document every time you look for a certain theme, the use of indexes is imperative.

System Characteristics

In a TextServer system, textual data is stored in the database using the LONG data type. The characteristics of this type of system are similar to those of the BLOB system. The data in the documents is stored in an unstructured state and has the following characteristics:

- ◆ **Significant size.** Each record is usually significant in size. Although the size depends on your particular documents, having rows of several hundred kilobytes or more than a megabyte is not uncommon.

- ◆ **Binary data.** The documents may contain ASCII data or may be in a binary format.
- ◆ **Compression.** Because the data may be binary and large, it may or may not be stored in a compressed state. This depends on the database and the application.

As you can see, many of the characteristics of a TextServer system are similar to those of the BLOB system. As with the BLOB system, the TextServer system has some unique data access patterns.

This type of system has one or more tables with records of a megabyte or more. When this data is accessed, there is a continuous data access to a single record. In the other types of database applications you have seen, it is unlikely that a single record is larger than a data block; with TextServer data, you are almost guaranteed that a single row spans multiple data blocks.

Design and Tuning Hints

Much of the design of the TextServer system is similar to that of the BLOB system: you try to maximize I/O to the data files. However, because a TextServer system is more likely than a BLOB system to have frequent updates to the database, the redo log files and archive log files are more important.

Design

The following system layout is based on the use of a disk array. You can easily convert this information for use with traditional Oracle disk-striping techniques. Here is the minimum configuration for a TextServer system:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance.
Redo log (1+)	The redo logs should be on their own disk volume because this type of system can see moderate to heavy insert and update activity. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance degrades during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1)	The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary.
Data and index (1+)	Because you always get concurrent access to disks in a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index files is determined by the amount of random I/O your user community generates.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. As with all systems, make sure that there is a sufficient number of disk drives to handle the load your users generate.

Tuning

Carefully analyze the following areas to see whether you should adjust these parameters:

- ◆ **DB_BLOCK_BUFFERS.** Because such large amounts of data are accessed either frequently or infrequently, it is hard to determine whether the SGA will be effective for caching data. The data cache may be effective because some updates and inserts may occur.
- ◆ **Library cache.** Check the library cache to see whether the shared pool is large enough. If necessary, increase the shared pool size.
- ◆ **Multiblock reads.** Because large amounts of data are accessed sequentially, the size of the multiblock read is important. The number of blocks read in a multiblock read is specified by the Oracle initialization parameter `DB_FILE_MULTIBLOCK_READ_COUNT`. This value, multiplied by the `DB_BLOCK_SIZE` parameter, results in the size of the I/Os. A good value for this initialization parameter is 64K or larger; try setting this parameter as high as Oracle allows (the value depends on the OS).
- ◆ **Data dictionary cache.** Check the efficiency of the data dictionary cache to see whether the shared pool size must be increased.
- ◆ **Rollback segments.** Because some updates are involved in this type of system, you may have to add rollback segments. Try using large rollback segments for this type of system.
- ◆ **Latch contention.** Latch contention should not be much of an issue unless you have an extreme amount of updates or inserts.
- ◆ **Checkpoints.** As with latches, I don't think you have to worry about checkpoints.
- ◆ **Archiving.** As with the other redo log-related parameters, archiving performance is usually not a problem in a TextServer system.

The area that probably requires the most attention is the I/O subsystem. By optimizing the I/O subsystem, you may see the best performance increases.

Enhancements

Some additional enhancements may improve the performance of your system. Listed here are some of the items that can offer performance improvement; also listed are a few items that do not improve performance. Each item is accompanied by an explanation.

- ◆ **Block size.** With a TextServer system, increasing the database block size can greatly improve performance. Because many of the queries access the database in a sequential manner (even though concurrent queries randomize this), having a larger block size brings more of the rows into the SGA at once. Having these additional rows in the SGA benefits you because you will be using them.

- ◆ **Clusters and hash clusters.** Clusters and hash clusters are not of any benefit in this type of system.
- ◆ **Indexes.** By using the index method provided with TextServer and ConText, you should see a significant benefit in searches and theme searches.
- ◆ **Parallel Query option.** The Oracle Parallel Query option is probably not of much use in this type of system.

Many of these enhancements can help performance. The specific effect on your system will vary, but the information presented here works well for most cases.

Review of the TextServer System

The Oracle TextServer 3.0 system is designed to assist with the storage, retrieval, and processing of textual data. TextServer 3.0 works with text data stored in an Oracle database and data external to the database. Using TextServer 3.0, you can not only store and retrieve text, you can perform additional functions such as indexing and textual search functions.

The data in the TextServer database is stored in the LONG data format. In most respects, this data is treated similarly to data in the BLOB system. But because the TextServer system sees more insert and update activity than a BLOB system, I recommend isolating the redo logs (a technique usually unnecessary in BLOB systems). When you take advantage of TextServer 3.0 and ConText, document storage and retrieval is more efficient as are other text-related functions.

Whether or not TextServer and ConText can be useful in your configuration depends on the amount of text processing you do. It is possible that the use of TextServer and ConText can greatly improve your efficiency.

Oracle Office Systems

The Oracle Office system is designed to provide office services such as messaging and scheduling. Oracle Office is a client/server application that uses the Oracle RDBMS system on the servers. The Oracle Office system also uses a replication method to duplicate data throughout the system.

The Oracle Office system consists of the following components:

- ◆ The Oracle Office client: the component that sits in the individual workstation.
- ◆ The Oracle Office server: the component that runs on the server.
- ◆ The Oracle RDBMS: the database used to maintain the data used by Oracle Office.

- ◆ SQL*Net: the communications link that eliminates the need for any specific network protocol.
- ◆ Other components: these include the gateway software, the Oracle Office Manager, and an API that lets you program your own interface to Oracle Office.

Oracle Office is a complex suite of components that can assist you in your day-to-day operations by providing general office services. In the scope of this book, I address only the Oracle Office server components and RDBMS.

System Characteristics

The Oracle Office system is designed to service many individual users simultaneously. Your office environment may consist of just a few or hundreds of clients accessing the same server. Because of this variety, the system appears very similar to the OLTP system in both characteristics and function. Some of these characteristics are listed here:

- ◆ **Simultaneous access to data.** Many users each access their messages, schedules, calendar, and directory services.
- ◆ **Data is accessed with both reads and writes.** Not only do users check messages, they continually produce new ones.
- ◆ **The database tends to grow.** As time passes, more and more data is saved in the system, creating a need for more space in the database.
- ◆ **Large number of users.** Although the number of users depends on your installation, typically, the entire corporation is connected to these types of services.
- ◆ **Response-time constraints.** As with the OLTP system, servicing online users requires that certain response times must be achieved. Response time can be a critical component of the system design.
- ◆ **Continuous uptime.** An office system may need to be in operation 24 hours per day, 7 days a week without any downtime. Special considerations must be made for fault-tolerant components or backup systems.

These characteristics may vary slightly based on your particular configuration and the number of users you service. Some components (such as the uptime requirement) can require additional hardware resources to provide additional fault tolerance.

Design and Tuning Hints

Because the system operates in a manner similar to that of a traditional OLTP system, many of the design and tuning parameters are similar to those of the OLTP system. The goals are to achieve good response times and a high level of fault tolerance.

Design

The layout for the office system is almost identical to that of the OLTP system except that you don't have as much control over the contents of the database tables themselves. A minimal configuration should look something like this:

Element (# of Disks)	Comments
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. Mirror this disk for additional fault tolerance.
Redo log (1+)	You should have at least one log volume. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance degrades during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary.
Data and index (1+)	Because you always get concurrent access to disks with a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates. If it is within your budget, I recommend mirroring the data and index files because of the uptime criterion usually associated with the office system.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks. From Chapter 14, “Advanced Disk I/O Concepts,” remember that you can only push a disk drive to a maximum random I/O rate.

Tuning

The office system is tuned almost identically to the OLTP system. The data access patterns that determine the tuning and design are similar in both systems. Carefully analyze these things to determine whether adjustment to these parameters is necessary:

- ◆ **DB_BLOCK_BUFFERS.** Block buffers are probably the most critical area in the office system. Having an insufficient number of block buffers results in higher-than-normal I/O rates and possibly an I/O bottleneck.

- ◆ **Library cache.** Check The V\$LIBRARYCACHE table, which contains statistics on how well you are using the library cache. You should be able to reduce the library cache misses by increasing the size of the shared pool.
- ◆ **Open cursors.** You may have to increase the number of cursors available for a session by increasing the Oracle parameter OPEN_CURSORS.
- ◆ **Cursor space for time.** If you have plenty of memory, you may be able to speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR_SPACE_FOR_TIME to TRUE.
- ◆ **Spin counts.** By tuning the Oracle initialization parameter SPIN_COUNT to enable spinning on resources (that is, the process spins instead of going to sleep while waiting for a resource to become available), you may see improved performance. SPIN_COUNT is useful when you have multiple CPUs and short-running transactions.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, look at the dynamic performance table V\$ROWCACHE. If necessary, you may have to increase the shared pool size.
- ◆ **Rollback contention.** You can tell if you are seeing contention on rollback segments by looking at the dynamic performance table V\$WAITSTAT. If the number is high, you need more rollback segments.
- ◆ **Use many rollback segments.** The rollback segments for an office system may have to be a little larger than the rollback segments traditionally used for OLTP.
- ◆ **Latch contention.** Latch contention can be determined by examining the dynamic performance table V\$LATCH. If there is a large amount of contention, you may need to take action as described in Chapter 9, “Oracle Instance Tuning.”
- ◆ **Checkpoints.** If the checkpoint processes are too hard on the system, you may have to adjust the duration and effect of the checkpoint.
- ◆ **Archiving.** You may have to adjust the effect archiving has on your system by changing the speed of the archive process.

These are some of the areas you should pay particular attention to when tuning a system used primarily for the Oracle Office. The areas that require the most attention are I/O and memory because they are so closely related. By optimizing the use of memory, you can reduce I/Os (which are probably running near the limitations of the hardware).

Review of the Oracle Office System

This section compared the characteristics of the Oracle Office system with those of the OLTP system. Because the two systems are similar, the layout of the office system and its tuning parameters are very similar to those of the OLTP system. Although the other components of the office system use additional CPU resources, they do not adversely affect the performance of the system.

The Oracle Office system is designed to provide office services such as messaging and scheduling. Oracle Office is a client/server application that uses the Oracle RDBMS system on the system servers. The Oracle Office system also uses a replication method to replicate data throughout the system.

If you are a user of the Oracle Office system, be sure to use the monitoring facility that comes with the Office package. It can help you find problem areas before they become critical.

WebServer Systems

The World Wide Web (WWW) encompasses a large amount of information, distributed in a common format among computer systems. This information is made up of documents, or *Web pages*. These Web pages are made up of text, images, and hypertext that links one Web page to another Web page.

In the few years since the WWW has been around, its popularity has grown at an amazing rate. It is very rare to see a large or medium sized company without its own business Web page. The World Wide Web is based on a set of open standards that allow all computer hardware and OS vendors—as well as Oracle—to participate in the Web.

It is only the Web's presentation system that is standardized. Where and how you store your data is up to you. The Oracle WebServer product is an add-on to the Oracle RDBMS that assists in these tasks. Using Oracle WebServer, you can store the data in a database for quick and personalized retrieval.

The Oracle World Wide Web interface kit as well as the WebServer system can help you use the power of Oracle with the unlimited potential of the Web.

System Characteristics

In general, the Oracle WebServer system has some characteristics of both OLTP and TextServer systems. It also has some unique characteristics:

- ◆ **Concurrent online users.** These users perform different activities against the same database.
- ◆ **Large amounts of text and image data.** The amount of data depends on your particular installation, but the type of data is primarily text and graphics.
- ◆ **Random data access.** The transactions tend to cause significant amounts of random I/O on the system.
- ◆ **Heavy network load.** The system is designed to serve users on the Internet. Depending on your traffic patterns, your network load can be quite high.

The data is accessed in a very random manner, as it is in an OLTP system. However, the WebServer access tends to be of a read-only nature. Currently, very few applications allow writing into a WWW site, but this is slowly changing.

Design and Tuning Hints

Because the Web system tends to have characteristics of both the OLTP and the TextServer system, the design of your system should take into account both characteristics. In designing a WebServer system, use the following goals to achieve optimal performance:

- ◆ **Isolate sequential I/Os.** Most of the time spent reading from or writing to the disk is spent seeking to where the data is located. If you can reduce seeks, you can achieve more I/Os per second.
- ◆ **Spread out random I/Os.** Random I/Os have a maximum rate per drive. By spreading the I/Os out across many drives, you increase the overall rate.
- ◆ **Avoid paging and swapping.** Any time the system pages or swaps, performance is severely degraded. Avoid this at all costs.

Design

The main goals in designing the physical data layout of a WebServer system are to provide balanced I/Os across all the disks that are randomly accessed and to isolate the sequential I/Os. In a WebServer system, you know that data is generally accessed in random fashion, but some of the data rows may be very large, spanning several blocks.

For most systems, you rely on the number of updates to determine whether it is worth it to separate the redo logs from the data volumes to take advantage of the sequential nature of its I/Os. For a WebServer system, this is a fairly hard call; without knowing the specific nature of your WebServer system, it is difficult to make a recommendation.

TIP: Without knowing the specifics of your system, I have to recommend that you separate the redo logs onto their own RAID-1 volume. Even if you are not performing a large number of updates, doing so certainly won't hurt your system's performance.

As with all transaction types, it is important to have a sufficient number of disk drives to handle the I/O generated by these transactions. It is important to monitor the system's I/O rates to ensure that the I/O subsystem is not overloaded during peak times.

The layout for this type of system looks more like an OLTP system than anything else. Because of the unknown nature of the log activity associated with this type of system, you may be able to minimize the size of the log volume and share the archive log volume with the OS or other disk volumes. A minimal configuration should look something like this:

<i>Element (# of Disks)</i>	<i>Comments</i>
System (1+)	The system disk is used for the operating system, swap file (if applicable), user files, and Oracle binaries. If possible, mirror this disk for additional fault tolerance. You can put the archive log files here if archiving is infrequent.
Redo log (1+)	You should have at least one log volume. This volume should be made up of at least two physical disks using RAID-1. By using only one log volume, performance is degraded during archiving because the sequential nature of the log writes is disrupted.
Archive logs (1+)	The number of disks needed for the archive log files is determined by the amount of data you have to archive. This data can be written to tape as necessary. If the amount and frequency of archiving is small, you can use the OS volume for archive logs.
Data and index (1+)	Because you always get concurrent access to disks with a disk array, it is not necessary to split the data and indexes into separate volumes. The number of disks you need for data and index is determined by the amount of random I/O your user community generates.

Both the data files and the indexes should be striped over as many disk drives as necessary to achieve optimal I/O rates on those disks.

I recommend taking advantage of a disk array rather than relying on Oracle striping. A hardware disk array provides you with the highest level of performance and has the added benefit of fault tolerance, if you need it.

Because of the nature of the WebServer, you may require a lot of fault tolerance. If your update activity is low, you may benefit from the use of RAID-5. Remember that RAID-5 has a high overhead for writes, but not for reads.

Tuning

The design of the WebServer system tends to follow that of the OLTP system, with the exception of the database block size, which looks more like that of the DSS system. When tuning a WebServer system, analyze the system to see whether adjustments to these parameters are necessary:

- ◆ **DB_BLOCK_BUFFERS.** Block buffers are probably the most critical area in the WebServer system. Having enough block buffers cuts down on the number of I/Os—and “hot” Web pages may get a very good cache-hit rate.

- ◆ **Library cache.** Check the V\$LIBRARYCACHE table, which contains statistics on how well you are using the library cache. If necessary, you may have to increase the size of the shared pool.
- ◆ **Open cursors.** You may have to increase the number of cursors available for a session by increasing the Oracle parameter OPEN_CURSORS.
- ◆ **Cursor space for time.** If you have plenty of memory, you can speed access to the shared SQL areas by setting the Oracle initialization parameter CURSOR_SPACE_FOR_TIME to TRUE.
- ◆ **Spin counts.** By tuning the Oracle initialization parameter SPIN_COUNT to enable spinning on resources (that is, the process spins instead of going to sleep while waiting for a resource to become available), you may see improved performance. SPIN_COUNT is useful when you have multiple CPUs and short-running transactions.
- ◆ **Data dictionary cache.** To check the efficiency of the data dictionary cache, look at the dynamic performance table V\$ROWCACHE. If necessary, you may have to increase the shared pool size.
- ◆ **Rollback contention.** Rollback contention should not be an issue in the WebServer system.
- ◆ **Latch contention.** Latch contention should not be much of an issue either.
- ◆ **Checkpoints.** As with rollbacks segments and latches, I don't think that checkpoints will be an issue.
- ◆ **Archiving.** By now, you should be convinced that you should always run with archiving enabled. By having a sufficiently large redo log, you may be able to put off archiving until off hours, especially if the number of updates is low.

As you can see, most of these recommendations are very similar to those given for the OLTP system.

Enhancements

You may want to make some additional enhancements to improve the performance of the WebServer system:

- ◆ **Block size.** Unlike the OLTP system, the WebServer system may benefit from a larger block size because of the large size of some of the text and image rows. I recommend trying a block size of 4,096 or 8,192.
- ◆ **Clusters.** I don't think that clusters will be of benefit, but your particular installation may be able to take advantage of them.
- ◆ **Hash clusters.** I don't think that hash clusters can benefit this type of system.
- ◆ **Multiblock reads.** Because some large amounts of data are read, multiblock reads can be beneficial. The value for multiblock reads should be 64K.

- ◆ **Parallel Query option.** The Oracle Parallel Query option will probably not help significantly in this type of application because there are many different user processes with relatively small queries.
- ◆ **Parallel Server option.** In this type of environment, you may be able to take advantage of both the performance and fault-tolerant features of the Oracle Parallel Server option.

Many of these enhancements can help your performance. The specific effect on your system will vary, but the information given here works well for most cases.

Review of the WebServer System

The WebServer system has the characteristics of both the OLTP and the TextServer systems. By considering both sets of characteristics and looking at the data access patterns, you can design the system to work optimally for both environments. Of course, you may have to make some compromises to accommodate both system types, but these may be minor.

As with the OLTP system, you must be concerned with the user response times in the WebServer system. Whether or not there are significant updates is determined solely by your configuration. Although these systems may not be considered mission critical, uptime may be an issue.

Summary

This chapter looked at several miscellaneous configurations and how to optimize them. You looked at the specific application type and contrasted it to another type of system or set of systems with which you are familiar. Using the general tuning guidelines listed in the preceding chapters and the system-specific guidelines presented in this chapter, you have a reasonable starting point from which to build the system.

From this starting point, you can run some test applications to further characterize the data access patterns and use of Oracle resources in your system. Then you can determine future changes that might benefit the performance of the system.

Part IV

Tuning SQL

Chapter 24 What Is a Well-Tuned SQL Statement?

- 25** Using EXPLAIN PLAN and SQL Trace
- 26** Tuning SQL Statements
- 27** Using the Oracle Optimizer
- 28** Using Procedures, Functions, and Packages
- 29** Providing for Data Integrity and Triggers
- 30** Using Hints
- 31** Introducing SQL Development Tools
- 32** Miscellaneous SQL Topics

In Part III of this book, you learned about the various ways to tune for specific application types. You learned not only how to tune the RDBMS and the server operating system, but also how to lay out the data to take advantage of data access patterns. You examined some specific operations and configurations such as the Oracle Parallel Server system and optimal backup and recovery. These examples and the information in Part II, “Tuning the Server,” should give you a good idea how to optimize the Oracle server system.

Part IV leaves the Oracle server and starts looking at the client side of the system. The following chapters analyze SQL statements and determine how these statements can be optimized, both for efficiency and to reduce unnecessary processing on the server. Part V, “Tuning the Client,” finishes the discussion of the client side of the system by looking at specific ways to optimize the client itself.

Tuning and optimizing the server is very important but cannot make up for a badly designed application. The server is designed to optimally process the SQL requests and return the data that is requested. Part II of this book showed you how to tune this process. It is possible, however for incorrectly designed SQL statements to scan massive amounts of unnecessary data and bypass well-designed indexes.

In Part IV, you learn how to determine whether your SQL statements are performing unnecessary functions and causing unwanted data to be returned to your client system. You learn how to use Oracle tools such as SQL Trace and EXPLAIN PLAN to determine whether your SQL statements are inefficient. You look at how to take advantage of the Oracle optimizer and how to use procedures and packages to best use the shared SQL area of the shared pool. Finally, you look at some products available to help you debug and optimize SQL statements.

When you finish the chapters in this part of the book, you should be able to determine the best way to form an SQL statement. You will be able to take advantage of features such as the Oracle Parallel Query option and to make best use of indexes, clusters, and hash clusters. You should also be able to use the SQL Trace and EXPLAIN PLAN utilities and analyze SQL statements with them.

Chapter 24

What Is a Well-Tuned SQL Statement?

This chapter begins to look at ways to produce a well-tuned application. The key to having an optimized application is at the heart of the application: the SQL statements themselves. Optimizing your SQL statements and reducing unnecessary overhead will result in an optimized application.

An optimized application together with the tuned and optimized RDBMS server will provide a well-balanced, highly tuned system. Because users are mainly interested in response times, having both a well-tuned application and an optimized server is essential. To get an optimized application, you must start with an optimized SQL statement.

So, what *is* a well-tuned, optimized SQL statement? Here's a list of some of the characteristics of a well-tuned SQL statement:

- ◆ **Makes efficient use of RDBMS features.** The well-tuned SQL statement uses indexes or hashing as available. If possible, the application should also take advantage of features such as array processing and discrete transactions.
- ◆ **Uses PL/SQL to improve performance.** PL/SQL allows blocks of statements to be sent to the Oracle server at one time. If you don't use PL/SQL, you must send each statement individually.
- ◆ **Uses stored procedures.** By using stored procedures, you reduce the amount of data that must be sent across the network and increase the chance that the statement may already be parsed in the shared SQL area.
- ◆ **Uses packages.** Packages increase performance because the entire package is loaded when the package is called for the first time.
- ◆ **Uses cached sequences to generate primary key values.** This improves the performance of key generation and makes it unnecessary to generate the key in the application.
- ◆ **Makes efficient use of space.** The SQL statement uses the VARCHAR2 data type instead of CHAR, when possible, to avoid unnecessary blank padding.
- ◆ **Uses hints where necessary.** A well-tuned SQL statement uses hints where appropriate to allow the programmer's understanding of the SQL statement and the database design to override Oracle's choice of optimization method.

These attributes in conjunction with your own specific attributes make a well-tuned SQL statement in your configuration. The properly tuned SQL statement avoids unnecessary functions and executes with the minimum amount of resources necessary to perform its function.

How To Identify Badly Formed SQL Statements

Badly tuned SQL statements tend to access the database in a very inefficient way, causing unnecessary amounts of data to be scanned and transferred across the network. Badly tuned statements can cause a well-tuned server to expend large amounts of unnecessary processing power and I/O resources.

You can identify badly tuned SQL statements with the Oracle EXPLAIN PLAN command and SQL Trace facility, as described in Chapter 25, "Using EXPLAIN PLAN and SQL Trace." Some of the attributes of a badly tuned SQL statement are listed here:

- ◆ Indexes are not used. If a query is not properly formed, you may bypass an index that could be used to reduce I/O and CPU processing.

- ◆ Hashing is bypassed. If a hashed cluster is improperly accessed, performance could be severely degraded.
- ◆ Unnecessary table scans are performed. If the SQL statement is improperly formed, you may be doing unnecessary table scans.
- ◆ Unnecessary amounts of data are returned. This is an undue burden not only on the network but on the application as well.

These attributes should alert you to the fact that the SQL statements are not optimally tuned for the task being done. If your SQL statement exhibits any of these characteristics, you should make some correction to the statement. The remaining chapters in Part IV of this book, “Tuning SQL,” explain how to correct these problems.

Transaction Processing

Before looking at specific SQL statements, this section analyzes how a transaction occurs in Oracle and then looks in more detail at how the SQL statement is actually parsed and the execution plan formed. Remember that a *transaction* is a logical group of work consisting of one or many SQL statements and ending with a commit or a rollback. In a typical transaction, the following steps are executed (see Figure 24.1):

A. Application Connection

1. The application processes the user input and creates a connection to the server through SQL*Net.
2. The server picks up the connection request and creates a server process on behalf of the user.

The application-connection process occurs only when the application is signing on. The application does not have to connect each time a statement is processed.

B. Application Processing

1. The user executes an SQL statement and commits the transaction. For example, the user changes the value of a row in a table.
2. The server process takes this SQL statement and checks the shared pool to see whether there is a shared SQL area that has this identical SQL statement. If it finds an identical shared SQL area, the server process checks to see whether the user has access privileges to the data and uses the shared SQL area to process the request. If a shared SQL area is not found, a new shared SQL area is allocated, the statement is parsed, and it is finally executed.
3. The server process retrieves the data from the SGA (if present) or retrieves it from the data file into the SGA.
4. The server process modifies the data in the SGA. Remember that the server processes can only read from the data files.

5. The LGWR process writes out the redo information. Not until this redo information has been written to the log is the statement considered committed. At some later time, the DBWR process writes the modified blocks to permanent storage.
6. If the transaction is successful, a completion code is returned across the network to the client process. If a failure occurs, an error message is returned.
7. Return to phase B, “Application Processing,” and submit more transactions until you are finished and want to exit the application.

The Application Processing phase is repeated indefinitely until the user is finished with this particular application and exits the application.

NOTE: A transaction is not considered committed until the write to the redo log file has been completed. This arrangement ensures that a committed transaction is recoverable in the event of a system failure. When a transaction has been committed and the redo entry has been written, the transaction is considered finished.

C. Application Termination

1. The application logs off the RDBMS. This event signals Oracle that all the associated resources can be deallocated.
2. The Oracle PMON process makes sure that the server process has been terminated.
3. All resources are released. Any memory resources the application has allocated are released.

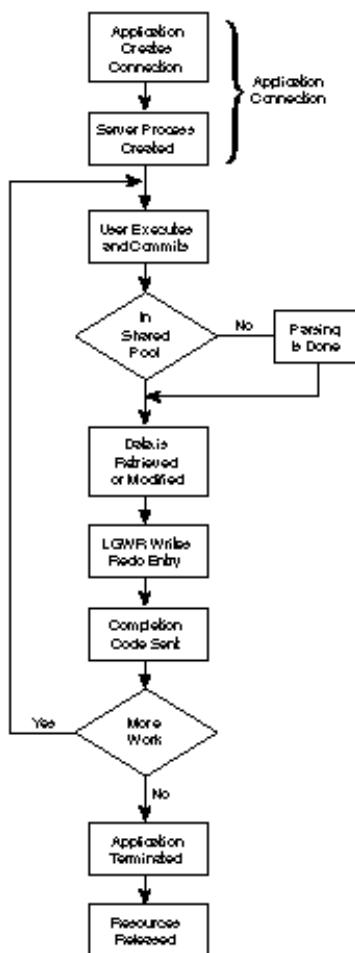
While this process is occurring, the Oracle background processes are doing their jobs keeping the system running smoothly. Keep in mind that your application is being processed, hundreds of other users may be doing similar tasks. It is Oracle’s job to keep the system in a consistent state, managing contention and locking and performing at the necessary rate.

Even though your application may have modified some data in the database, that data may not yet be written to the data files. It may be some time later that the DBWR process writes those changes out to permanent storage.

With this overview of how the application is processed, you are ready to focus on what happens in step 2 of phase B: how the SQL statement is parsed and the execution plan is formed.

Figure 24.1

A flowchart showing how an application is processed.



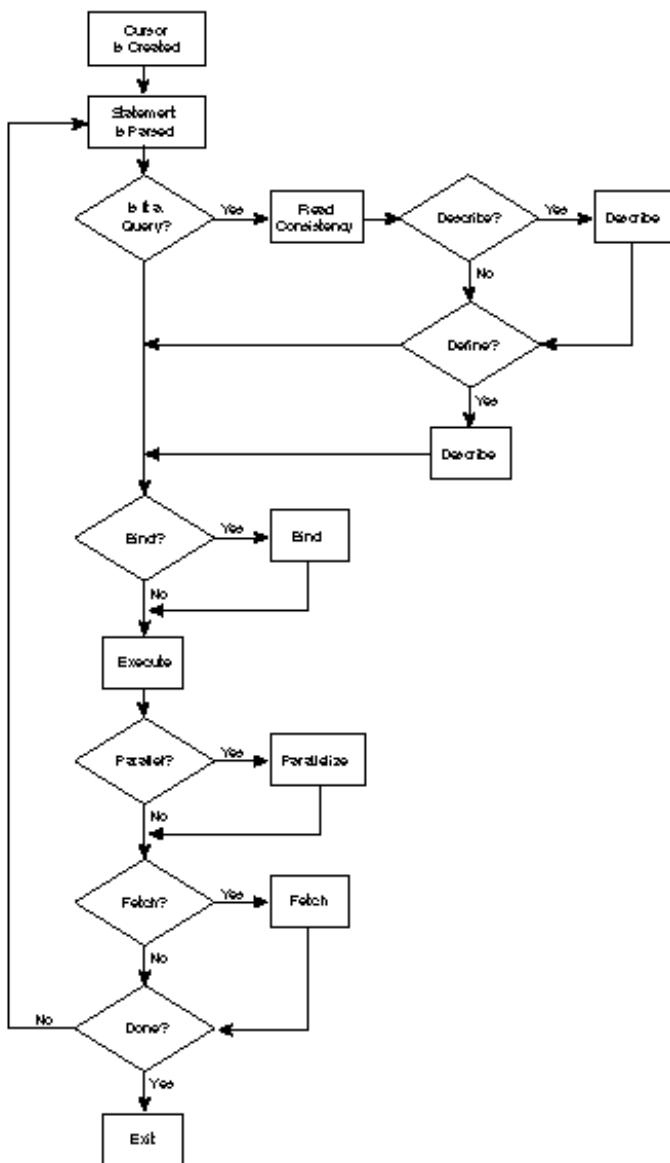
SQL Statement Processing

By understanding how Oracle processes SQL statements, you can have a better understanding of how to optimize these statements. The following sections look at the SQL statement parsing process and how an execution plan is formed. For each SQL statement that is executed, several steps occur (see Figure 24.2):

1. A cursor is created.
2. The statement is parsed, if it is not already in the shared pool.
3. Any query in the statement is processed.
4. Variables are bound.
5. The statement is executed.
6. If possible, the statement is parallelized.
7. Rows to be returned are fetched.

Figure 24.2

A flowchart showing how an SQL statement is processed.



Cursor Creation

Each time an SQL statement is executed, a cursor is automatically created on behalf of the statement. If you want, you can declare the cursor manually. Remember that a *cursor* is a handle to a specific private SQL area. You can think of a cursor as a pointer to, or the name of, a particular area of memory associated with an SQL statement.

Statement Parsing

Once the cursor is created, a determination is made about whether the SQL statement is already present in the shared SQL area in the shared pool. If the SQL statement has already been parsed and is in the shared pool, there is no further need for parsing, and the execution of the SQL statement continues. By using stored procedures or by carefully crafting SQL statements to be identical, you stand a good chance that those statements will be in the shared SQL area, already parsed.

For an SQL statement to take advantage of SQL or PL/SQL statements that may have already been parsed, the following criteria must be met:

- ◆ The text of the SQL statement must be identical to the SQL statement that has already been parsed. This includes whitespaces.
- ◆ Reference to schema objects in the SQL statements must resolve to the same object.
- ◆ Bind variables must match the same name and data type.
- ◆ The SQL statements must be optimized using the same approach; in the case of the cost-based approach, the same optimization goal must be used.

You may think that these conditions make it difficult to take advantage of the shared SQL areas. In fact, users sharing the same application code meet these criteria quite easily. It is to the advantage of the application developer to use the same SQL statements to access the same data, ensuring that SQL statements within the application can also take advantage of the shared SQL areas.

TIP: Using stored procedures whenever possible guarantees that the same shared PL/SQL area is used. Another advantage is that stored procedures are stored in a parsed form, eliminating runtime parsing altogether.

Standardizing on naming conventions for bind variables and spacing conventions for SQL and PL/SQL statements also increases the likelihood of reusing shared SQL statements.

The V\$LIBRARYCACHE table contains statistics on how well you are using the library cache. The important columns to view in this table are PINS and RELOADS. The PINS column contains the number of times the item in the library cache was executed. The RELOADS column contains the number of times the library cache missed and the library object was reloaded. A few number of reloads relative to the number of executions indicates a high cache-hit rate for shared SQL statements.

If the statement is not in the shared pool, the following steps are executed to parse the SQL statement:

1. **The statement is validated.** The SQL statement must be verified as a valid statement.
2. **The data is validated.** The data dictionary lookups are performed to verify that the table and column definitions are correct.
3. **Locks are allocated.** Parse locks must be acquired to make sure that object definitions don't change during the execution of the parsing.
4. **Privileges are verified.** Oracle validates that the user has permission to use the schema objects being accessed.
5. **The execution plan is determined.** The optimal execution plan is determined based on several factors, including optimization plans, hints, and database analysis.
6. **The statement is loaded into the shared SQL area.** Once the execution plan has been determined, the statement is loaded into the shared SQL area.
7. **The distributed statement is routed.** If the statement is used as a distributed transaction, all or part of the statement is routed to the other nodes involved in this statement.

As you can see from the number of steps that must be executed, it is important to try to keep the SQL statements in the shared pool to avoid the parsing phase of the execution process.

Query Processing

Queries are handled differently than other SQL statements because queries return data as the result of the statement. Other SQL statements need only return a return code that indicates success or failure. In addition to the other steps that must be executed, queries may require the following additional functions:

- ◆ **Read consistency.** Because you may be executing several statements that take considerable time, it is important that the data remain consistent through the lifetime of the query.
- ◆ **Use of temporary segments.** Because queries may perform additional functions such as joins, ORDER BYs, sorts, and so on, it may be necessary to use temporary segments.
- ◆ **Describe the results (optional).** This phase is necessary if the characteristics of the query's results are not known (for example, with an interactive query, the data types of the results must be determined before the results can be returned).
- ◆ **Output definition (optional).** If the output location, size, and variable data types are defined, it may be necessary for Oracle to perform data conversions.

Only for queries are the preceding functions necessary in addition to the other SQL statement processing.

Bind Variables

Variables must be defined for statement to be processed. The program must specify to Oracle the address of the variable before Oracle can *bind* that variable. Because the binding is done by reference, you do not have to rebind a variable before reexecuting the statement; simply changing its value is sufficient.

You must supply the data type and length of each variable you bind to Oracle unless these data types or lengths are implied or defaulted.

Statement Execution

Once the statement has been parsed and the variables have been defined, the statement is executed. In array processing, the execution step may happen many times. Any necessary locks are applied before the execution of the statement.

Parallelization

If the Oracle Parallel Query option is used and the statement being executed is parallelizable, the following steps take place:

1. The query coordinator determines which operations can be performed in parallel.
2. The query coordinator determines how many query servers to enlist.
3. The query coordinator enlists query servers that perform the query.
4. The query coordinator reassembles the resulting data and passes it back to the user.

The degree of parallelism is determined using the following order of precedence:

1. **Query hints.** User-defined hints included in the SQL statement have the highest precedence.
2. **Table definition.** The default degree of parallelism defined for the table is second in the order of precedence.
3. **Initialization parameters.** Finally, the Oracle initialization parameters are used to determine parallelism.

The processes that execute the query are taken from the set of query servers available in the query server pool. This number is specified by the Oracle initialization parameter `PARALLEL_MAX_SERVERS`.

Fetch Rows To Be Returned

If the statement is a query, the final step in the processing of the SQL statement involves fetching the returned data in a loop until all the requested data has been returned to the user process.

Review of SQL Statement Processing

By understanding the process that takes place when an SQL statement is executed, you can see the value in avoiding some of these steps.

Taking advantage of SQL statements that have already been parsed is one way to limit the amount of overhead associated with the processing of the statement.

Summary

Badly formed SQL statements can nullify any optimizations you have made to the client or the server. If the SQL statements require unnecessary table scans or unnecessary processing, an optimized server can be brought to a standstill. To have the most-optimal system possible, each component—the client, the SQL statements, the network, and the server—should be tuned to take maximum advantage of available resources. The best way to improve performance is to reduce unnecessary processing and I/O.

You can do this by designing your SQL statements to take advantage of such features as the shared pool (where the library cache contains the shared SQL area). By designing your application so that it uses the same SQL statements to access the same data, you stand a good chance of your statements being in the library cache.

If you have long-running queries that perform table scans, take advantage of the Parallel Query option. By using this feature, you can cut down on the time it takes to perform the query.

The use of other Oracle features such as stored procedures and packages can also maximize the effectiveness of your SQL statement processing and provide for more efficient coding. All these things can help improve the overall performance of your system.

The remaining chapters in Part IV of this book look into ways you can make your SQL statements more efficient—and how you can determine the efficiency of your SQL statements.

Of all the areas of the system that can be tuned, SQL statements offer the most potential. This is true not because SQL statement can be more effectively optimized, but because a badly tuned SQL statement can degrade performance drastically. As you will see, it is very important to have well-tuned SQL statements at the heart of your application.

Chapter 25

Using EXPLAIN PLAN and SQL Trace

Probably the best way to determine whether your SQL statements are properly optimized is by using the Oracle SQL Trace facility and the EXPLAIN PLAN command. You can use the SQL Trace facility and the Oracle program TKPROF, which is used to translate trace files, to trace production SQL statements, and gather statistics about those statements.

You use SQL Trace to gather information into a trace file; the Oracle program TKPROF formats the trace information into useful, understandable data.

The EXPLAIN PLAN command is used to display the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. By analyzing the execution plan the Oracle optimizer has chosen, and knowing your data and application, you should be able to determine whether the optimizer has chosen the correct execution plan for your application.

After using EXPLAIN PLAN, you can rewrite your SQL statements to take better advantage of such things as indexes and hash keys. By analyzing the output, you may be able to provide hints that the Oracle optimizer can use to take better advantage of your knowledge of your data. By using hints, you may be able to take better advantage of features such as the Oracle Parallel Query option.

By the end of this chapter, you should be able to execute SQL statements using both SQL Trace and EXPLAIN PLAN and be able to analyze the output from these statements. You should also understand the value of registering applications for later use when tracking performance problems. These Oracle options can greatly improve the stability and performance of your system.

SQL Trace

The SQL Trace facility and the Oracle program TKPROF are designed to give performance information about individual SQL statements. You can use this information to determine the characteristics of those statements.

You can enable SQL Trace for a session or for an entire instance. Of course, because this facility gathers an abundance of information about SQL statement functionality and performance, SQL Trace has an effect on the performance of the system. If you use SQL Trace on a single session, the effect is fairly minimal, but if you use SQL Trace on an entire instance, you will see a substantial effect on the performance of the system. Avoid running SQL Trace on an entire instance for this reason.

SQL Trace Initialization

Before you run SQL Trace, you must make sure that certain Oracle initialization parameters are set:

<i>Parameter</i>	<i>Description</i>
TIMED_STATISTICS	Setting TIMED_STATISTICS to TRUE enables SQL Trace and some of the dynamic performance tables to collect timed statistics such as CPU and elapsed times. Enabling timed statistics incurs significant overhead because most Oracle operations are now being timed; avoid this parameter except when necessary.

<i>Parameter</i>	<i>Description</i>
MAX_DUMP_FILE_SIZE	Specifies the maximum size of trace file dumps in OS blocks. Set this fairly low to avoid filling up the file system with trace files. If the SQL Trace output files are being truncated, increase this value.
USER_DUMP_DEST	This parameter specifies the destination for the trace file. The default destination is the same as for system dumps on your OS.

Controlling SQL Trace

You can enable the SQL Trace facility on a per-session basis or for the entire instance. The following sections explain how to enable and disable SQL Trace for both of these cases.

Enable SQL Trace for a Session

To enable SQL Trace for a session, use this Oracle command:

```
ALTER SESSION
SET SQL TRACE = TRUE;
```

Alternatively, you can use the Oracle procedure RDBMS_SESSION.SET_SQL_TRACE.

To enable SQL Trace for a session other than your own, you can use the Oracle procedure RDBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION with the arguments SID, Serial#, and TRUE. To determine the values for SID and Serial#, use the following SQL statements:

```
SQL> SELECT sid, serial#, osuser
  2  FROM v$session
  3  WHERE osuser = 'Ed Whalen';

  SID    SERIAL# OSUSER
  -----
  7        4 Ed Whalen
```

To turn SQL Trace on for that session, use the Oracle stored procedure as follows:

```
SQL> EXECUTE RDBMS_system.set_sql_trace_in_session(7,4,TRUE);
PL/SQL procedure successfully completed.
```

Disable SQL Trace for a Session

To disable SQL Trace for a session, use this Oracle command:

```
ALTER SESSION
SET SQL TRACE = FALSE;
```

The SQL Trace facility is also disabled when your session disconnects from Oracle.

To disable SQL Trace for a session other than your own, use the Oracle procedure `RDBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` with the arguments `SID`, `Serial#`, and `FALSE` as shown here:

```
SQL> EXECUTE RDBMS_system.set_sql_trace_in_session(7,4, FALSE);
PL/SQL procedure successfully completed.
```

Enable SQL Trace for an Instance

To enable SQL Trace for your instance, set the Oracle initialization parameter `SQL_TRACE` to `TRUE`. Doing so enables SQL Trace for all users of this instance for the duration of the instance.

Disable SQL Trace for an Instance

The SQL Trace facility cannot be disabled for the entire instance without shutting down the Oracle instance and setting the Oracle initialization parameter `SQL_TRACE` to `FALSE`. Alternatively, you can remove the parameter because its default value is `FALSE`.

When SQL Trace is enabled for the entire instance, it is still possible to disable it on a per-session basis. You can disable SQL Trace on a per-session basis with the SQL statement shown in the preceding section.

SQL Trace Functionality

Once SQL Trace is enabled, it gathers the following information:

- ◆ **Parse, execute, and fetch counts.** These counts can give you vital information about the efficiency of the SQL statements.
- ◆ **CPU and elapsed times.** This information can tell you which statements take the most time to execute.
- ◆ **Physical and logical reads.** This information can help you determine the effectiveness of the database buffer pool.
- ◆ **Number of rows processed.** This information can be used as an indication that more rows are being processed than you expected, thus indicating a problem.
- ◆ **Library cache misses.** This information can show you the effectiveness of the shared SQL area and how well you are reusing already parsed SQL statements.

SQL Trace puts this information into a trace file in an unreadable form. You then use the Oracle program `TKPROF` to format the trace information into useful, understandable data.

TKPROF Functionality

You use the Oracle program TKPROF to convert the SQL Trace information into data that is formatted to be understood by human beings. By specifying certain options, you can customize the output of TKPROF to some degree. TKPROF is invoked with the following syntax:

```
TKPROF inputfile outputfile [ Optional Parameters ]
```

In this syntax, *inputfile* is the data file generated by SQL Trace and *outputfile* is the name of the file to which you want the output of TKPROF to be written.

The optional parameters for KTPROF are as follows:

<i>Parameter</i>	<i>Description</i>
<code>EXPLAIN = <i>username/password</i></code>	This option automatically runs the EXPLAIN PLAN command and adds the execution plan to the output of SQL Trace.
<code>TABLE = <i>schema.table</i></code>	This option specifies the schema and table name of the temporary table TKPROF uses when processing the SQL Trace data. If this option is not specified, TKPROF creates, uses, and then deletes a temporary table.
<code>INSERT = <i>scriptfile</i></code>	Creates a file of name <i>scriptfile</i> that contains the SQL statements TKPROF uses for storing trace file statistics.
<code>SYS = [YES/NO]</code>	Determines whether SQL statements generated by user SYS or recursive SQL statements are listed.
<code>PRINT = <i>number</i></code>	Generates the output of only the first <i>number</i> of sorted SQL statements.
<code>RECORD = <i>recordfile</i></code>	Creates a file named <i>recordfile</i> that contains all recursive SQL statements.
<code>SORT = <i>sort_option</i></code>	This option sorts the output of SQL Trace in descending order based on the <i>sort_option</i> specified. The <i>sort_option</i> variable can be any of the following values:
PRSCNT	Sorted by parse count
PRSCPU	Sorted by CPU time spent parsing
PRSEL A	Sorted by elapsed time spent parsing

continues

<i>Parameter</i>	<i>Description</i>
PRSDSK	Sorted by number of physical reads from disk during parse
PRSQRY	Sorted by number of consistent mode block reads during parse
PRSCU	Sorted by number of current mode block reads during parse
EXECNT	Sorted by number of executes
EXECPU	Sorted by CPU time spent executing
EXEELA	Sorted by elapsed time spent executing
EXEDSK	Sorted by number of physical reads during execute
EXEQRY	Sorted by number of consistent mode block reads during execute
EXECU	Sorted by number of current mode block reads during execute
EXEROW	Sorted by number of rows processed during execute
EXEMIS	Sorted by number of library cache misses during execute
FCHCNT	Sorted by number of fetches
FCHCPU	Sorted by CPU time spent fetching
FCHELA	Sorted by elapsed time spent fetching
FCHDSK	Sorted by number of physical reads from disk during fetch
FCHQRY	Sorted by number of consistent mode block reads during fetch
FCHCU	Sorted by number of current mode block reads during fetch
FCHROW	Sorted by number of rows fetched

With these options, SQL Trace can provide an abundance of data that can help you analyze your SQL statements. The following section examines some of the data SQL Trace provides.

Interpreting SQL Trace

This section looks at some of the statistics available from SQL Trace and how to interpret them. For each SQL statement executed, SQL Trace provides the following information:

<i>Parameter</i>	<i>Description</i>
count	Number of times the OCI procedure was executed. (The OCI interface is the standard set of calls used to access the Oracle database.)
cpu	CPU time in seconds executing. This value is the amount of time Oracle used to process the statement.
elapsed	Elapsed time in seconds executing. This value is equivalent to the user's response time.
disk	Number of physical reads of buffers from disk. This value tells you how many reads actually missed the buffer cache and had to go to physical disk.
query	Number of buffers gotten for consistent read. This value represents the number of buffers retrieved in consistent mode. Consistent mode guarantees consistent reads throughout the transaction; it is used for most queries.
current	Number of buffers gotten in current mode (usually for update). In current mode, the data blocks gotten reflect the value at that instant in time.
rows	Number of rows processed by the fetch or execute call. This value gives you an idea of how many instructions have been executed.

By looking at each of these parameters, you can get an idea of how your SQL statements are being processed and which statements are taking the most time. By analyzing which statements are taking the longest, you may be able to find some inefficiencies you can correct.

The SQL Trace facility was enabled in a session by using this Oracle command:

```
EXECUTE RDBMS_system.set_sql_trace_in_session(7,3,TRUE);
```

In another session (with `SID = 7` and `Serial# = 3`), the following SQL statement was executed:

```
SELECT
    SUBSTR(dogname,1,20) "Dog Name",
    SUBSTR(description,1,20) "Breed",
    SUBSTR(owner,1,20) "Owner"
FROM
    dogs, breeds
WHERE
    dogs.breed = breeds.breed
ORDER BY
    dogs.breed;
```

The SQL Trace facility was later disabled using this Oracle command from the first session:

```
EXECUTE RDBMS_system.set_sql_trace_in_session(7,3, FALSE);
```

Following the execution of the SQL statements, the trace file was translated by running TKPROF as follows:

```
tkprof orclshad.trc trace.out sys=no explain=ed/ed
```

In this syntax, the following are true:

<i>orclshad. trc</i>	Trace file generated by SQL Trace
<i>trace. out</i>	Where I want the output to go
<code>sys=no</code>	Indicates that no SYS or recursive SQL statements should be printed
<code>explain=ed/ed</code>	Specifies that I also want to generate EXPLAIN PLAN output

TKPROF generated the output file shown in Listing 25.1.

Listing 25.1 The TKPROF Output File for a Sample Trace

```
TKPROF: Release 7.2.2.3.0 - Beta on Sun Nov 26 13:03:20 1995
```

```
Copyright Oracle Corporation 1979, 1994. All rights reserved.
```

```
Trace file: orclshad.trc
Sort options: default
```

```
*****
count      = number of times OCI procedure was executed
cpu        = cpu time in seconds executing
elapsed    = elapsed time in seconds executing
disk       = number of physical reads of buffers from disk
query      = number of buffers gotten for consistent read
current    = number of buffers gotten in current mode (usually for update)
rows       = number of rows processed by the fetch or execute call
*****
```

```
SELECT
    SUBSTR(dogname,1,20) "Dog Name",
    SUBSTR(breed_name,1,20) "Breed",
    SUBSTR(owner,1,20) "Owner"
```

```

FROM
  dogs, breeds
WHERE
  dogs.breed = breeds.breed
ORDER BY
  dogs.breed

call      count      cpu    elapsed      disk      query      current      rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.00      0.57          2          0          4          0
Execute       1      0.00      0.00          0          0          0          0
Fetch        2      0.00      0.08          2          2          6         25
-----  -----  -----  -----  -----  -----  -----  -----
total        4      0.00      0.65          4          2         10         25

Misses in library cache during parse: 1
Optimizer hint: CHOOSE
Parsing user id: 10 (ED)

```

Rows Execution Plan

```

0  SELECT STATEMENT  HINT: CHOOSE
60  MERGE JOIN
0   SORT (JOIN)
0   TABLE ACCESS (FULL) OF 'BREEDS'
0   SORT (JOIN)
0   TABLE ACCESS (FULL) OF 'DOGS'

```

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.57	2	0	4	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	2	0.00	0.08	2	2	6	25
total	4	0.00	0.65	4	2	10	25

Misses in library cache during parse: 1

```

1  user  SQL statements in session.
8  internal SQL statements in session.
9  SQL statements in session.
1  statement EXPLAINed in this session.

```

Trace file: orclshad.trc

Trace file compatibility: 7.02.01

Sort options: default

```

1  session in tracefile.
1  user  SQL statements in trace file.
8  internal SQL statements in trace file.
9  SQL statements in trace file.

```

continues

Listing 25.1 continued

```

6 unique SQL statements in trace file.
1 SQL statements EXPLAINed using schema:
  ED.prof$plan_table
    Default table was used.
    Table was created.
    Table was dropped.
109 lines in trace file.

```

Notice that there are three user SQL statements defined in the output from SQL Trace (Listing 25.1) but only one SQL statement is shown. This is because the Oracle commands used by the other session to enable SQL Trace are included in the output. For now, focus on the output related to the following SQL statements (the output follows the SQL statements):

```

SELECT
  SUBSTR(dogname,1,20) "Dog Name",
  SUBSTR(breed_name,1,20) "Breed",
  SUBSTR(owner,1,20) "Owner"
FROM
  dogs, breeds
WHERE
  dogs.breed = breeds.breed
ORDER BY
  dogs.breed

call      count        cpu      elapsed       disk      query     current      rows
-----  -----
Parse        1        0.00      0.57          2          0         4          0
Execute      1        0.00      0.00          0          0         0          0
Fetch        2        0.00      0.08          2          2         6         25
-----
total       4        0.00      0.65          4          2        10         25

Misses in library cache during parse: 1
Optimizer hint: CHOOSE
Parsing user id: 10  (ED)

```

From this output, you can see that the majority of the elapsed time was spent in the parse phase. The CPU time to execute this statement was less than 0.01 second (which is the resolution of timed statistics). The following chart describes these statistics in a little more detail.

<i>Parameter</i>	<i>Value and Meaning</i>
count	The count value is 1, 1, and 2 (Parse, Execute, and Fetch), indicating that the SQL statement executed an OCI call once during the parse, again in the execute phases, and twice during the fetch.
cpu	The CPU time spent in each of the phases was 0.0, 0.0, and 0.0 seconds (Parse, Execute, and Fetch). These minuscule times are the result of the tiny size of the database tables and the query. Your SQL statements will probably use much more CPU time.

<i>Parameter</i>	<i>Value and Meaning</i>
elapsed	The elapsed time. By far, the majority of the time executing this SQL statement is in the parse phase. However, this is not true for more complex statements and larger tables.
disk	Notice that it is only during the parse and fetch phases that there is any disk activity.
query	Because this SQL statement is a query, the fetches are done in consistent mode rather than current mode.
current	The reads for the parsing phase and some of the reads for the fetching phase were done in current mode. Reads are done in current mode whenever possible.
rows	The fetch phase processed 25 rows. Because this is a query operation, this value indicates how many rows were returned. In an UPDATE, INSERT, or DELETE operation, the rows parameter in the fetch phase would indicate how many rows were processed in each of these statements.

This information can give you valuable insight into how your queries are running. Although the output for the EXPLAIN PLAN command was also given here, it is described later in this chapter.

In addition to the preceding information about the SQL statements statistics, SQL Trace also provides the following information about library cache statistics and optimizer hints:

```
Misses in library cache during parse: 1
Optimizer hint: CHOOSE
Parsing user id: 10 (ED)
```

With this particular SQL statement, there was 1 library cache miss (which is quite high, considering that only one SQL statement was executed). The Optimizer hint: CHOOSE line indicates that the optimization method was left to Oracle to decide.

Also of importance are any recursive SQL statements that Oracle has to execute on behalf of the user's SQL statement. In this example, I used TKPROF with the parameter SYS=NO to prevent the reporting of recursive statements. I did that because I was manually turning SQL Trace on and off and did not want to show a lot of irrelevant output. I also knew that the sample SQL statement would not generate any recursive calls.

I recommend that you do **not** set SYS=NO unless you are in a similar situation. It is very useful to obtain an indication of any recursive calls being executed. Later in this chapter, you use EXPLAIN PLAN to get an idea of how the SQL statement has been executed.

Review of SQL Trace

The first part of this chapter looked at SQL Trace, a very powerful facility provided by Oracle for debugging SQL performance problems. SQL Trace can provide much valuable information you can use to debug many different types of performance problems.

SQL Trace provides valuable information on such things as these:

- ◆ Parse, execute, and fetch counts
- ◆ CPU and elapsed times
- ◆ Physical and logical reads
- ◆ Number of rows processed
- ◆ Library cache misses

The EXPLAIN PLAN Command

The EXPLAIN PLAN command shows you the execution plan that the Oracle optimizer has chosen for your SQL statements. With this information, you can determine whether the Oracle optimizer has chosen the correct execution plan based on your knowledge of the data and the application. You can also use EXPLAIN PLAN to determine whether any additional optimization should be done to your database (for example, the addition of an index or the use of a cluster).

The EXPLAIN PLAN command is used to display the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. After using EXPLAIN PLAN, you can rewrite your SQL statements and see whether the new SQL statement is better optimized than the original statement. By analyzing the output, you may be able to provide hints that the Oracle optimizer can use to take better advantage of the data (hints are described in Chapter 30, “Using Hints”). By using hints, you can take better advantage of features such as the Oracle Parallel Query option.

EXPLAIN PLAN Initialization

When you run SQL statements with the EXPLAIN PLAN command, the output of EXPLAIN PLAN is put into a table with the default name plan_table. You must create this table before you can run EXPLAIN PLAN. The table can be created in one of two ways:

- ◆ Using the UTLXPLAN.SQL script provided by Oracle.
- ◆ Creating the plan_table table by hand.

The plan_table table is define as follows:

```
SQL> describe plan_table
Name          Null?    Type
-----        -----
STATEMENT_ID           VARCHAR2(30)
TIMESTAMP             DATE
REMARKS               VARCHAR2(80)
OPERATION             VARCHAR2(30)
OPTIONS               VARCHAR2(30)
OBJECT_NODE            VARCHAR2(128)
OBJECT_OWNER            VARCHAR2(30)
OBJECT_NAME             VARCHAR2(30)
OBJECT_INSTANCE          NUMBER(38)
OBJECT_TYPE              VARCHAR2(30)
OPTIMIZER                VARCHAR2(255)
SEARCH_COLUMNS           NUMBER(38)
ID                     NUMBER(38)
PARENT_ID               NUMBER(38)
POSITION                 NUMBER(38)
OTHER                   LONG
```

You do not have to name the table plan_table. You can direct EXPLAIN PLAN to use a table of another name if you want.

Invoking EXPLAIN PLAN

Invoke the EXPLAIN PLAN command with the following Oracle command sequence:

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'Testing EXPLAIN PLAN'
  INTO plan_table
  FOR
    SQL Statement;
```

STATEMENT_ID should reflect the statement's function so that you can recognize it at a later time. The plan_table parameter is the name of the table you created as described in the preceding section. If the INTO clause is omitted, the command defaults to the name plan_table.

Here is an example of a completed command:

```
SQL> EXPLAIN PLAN
  2      SET STATEMENT_ID = 'Testing EXPLAIN PLAN'
  3      INTO plan_table
  4      FOR
  5          SELECT
  6              SUBSTR(dogname,1,20) "Dog Name",
  7              SUBSTR(breed_name,1,20) "Breed",
  8              SUBSTR(owner,1,20) "Owner"
  9          FROM
 10              dogs, breeds
 11          WHERE
 12              dogs.breed = breeds.breed
 13          ORDER BY
 14              dogs.breed;
```

Explained.

The results of the EXPLAIN PLAN are written into the table plan_table. The following section explains how to retrieve the information in that table.

Extracting EXPLAIN PLAN Results

The output of EXPLAIN PLAN is written to the table specified in the EXPLAIN PLAN command (by default, to the table named plan_table). You must extract this information in order to look at the results of EXPLAIN PLAN. The results can be displayed with a query such as this:

```
SELECT SUBSTR(LPAD(' ',2*(LEVEL-1))||operation,1,30)
||' '|SUBSTR(options,1,15)
||' '|SUBSTR(object_name,1,15)
||' '|SUBSTR(DECODE(id, 0, 'Cost = '||position),1,12)
"Statement Execution Plan",
SUBSTR(optimizer, 1, 10) "Optimizer"
FROM
    plan_table
START WITH
    id = 0 AND statement_id = 'Testing EXPLAIN PLAN'
CONNECT BY PRIOR
    id = parent_id
AND
    statement_id = 'Testing EXPLAIN PLAN';
```

This query results in the following output:

Statement Execution Plan	Optimizer

SELECT STATEMENT Cost =	CHOOSE
MERGE JOIN	
SORT JOIN	
TABLE ACCESS FULL BREEDS	
SORT JOIN	
TABLE ACCESS FULL DOGS	

6 rows selected.

If the optimizer had chosen a cost-based approach, the cost of the query would have been reflected in the first line of the optimization plan. Any features such as parallel query are also reflected here.

With this information, you can tell whether your SQL statements take advantage of indexes, clusters, or hash clusters. If you use EXPLAIN PLAN, you can see precisely how your SQL statement is being executed and what effect any changes you make to the SQL statements have on the execution plan. If you change your SQL statements to take advantage of an index or a cluster, for example, you can see an immediate improvement. EXPLAIN PLAN output is ideal for pointing out your execution plan and may indicate that where you thought you were taking advantage of an index, you actually were not.

Registering Applications

When you register an application, the name and the actions performed by that application are stored in the database to assist with debugging and performance tuning efforts. When an application is registered, its name and actions are recorded in the V\$SESSION and V\$SQLAREA views. This information can be used later to track problems.

To register an application, use the following procedures, available in the RDBMS_APPLICATION_INFO package:

<i>Procedure</i>	<i>Description</i>
SET_MODULE	Used to set the name of the module currently being run.
SET_ACTION	Used to set the name of a certain action currently being performed.
SET_CLIENT_INFO	Used to set up information for the client information field.
READ_MODULE	Reads the current values of the module and action fields for the current session.
READ_CLIENT_INFO	Reads the current client information field for the currently running session.

By registering the application, you can track many different parameters. Some of the values available through V\$SQLAREA are given here:

- ◆ Memory used
- ◆ Number of sorts
- ◆ Number of executions
- ◆ Number of loads
- ◆ Number of parse calls
- ◆ Number of disk reads
- ◆ Number of buffer gets
- ◆ Number of rows processed

These parameters can provide valuable information when you are trying to debug various modules within your application. The information is enhanced by the addition of actions, which can further identify sections of your application.

Summary

Determining whether your SQL statements are properly optimized can be as important as anything else you can do to tune your system. An improperly tuned SQL statement can nullify any work you have done to optimize the database system. A well-tuned server system that is handling hundreds or thousands of unnecessary SQL statements can be perceived to have poor performance when, in reality, there is just an abundance of excess work being done.

The Oracle SQL Trace facility and the EXPLAIN PLAN command can be valuable tools in debugging inefficient SQL code. The SQL Trace facility and its companion program TKPROF can give valuable information into such areas as these:

- ◆ Parse, execute, and fetch counts
- ◆ CPU and elapsed times
- ◆ Physical and logical reads
- ◆ Number of rows processed
- ◆ Library cache misses

The EXPLAIN PLAN command is used to display the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. By analyzing the execution plan the Oracle optimizer has chosen, and by knowing your data and application, you should be able to determine whether the optimizer has chosen the correct execution plan for your application. If you do not agree with the execution plan that has been chosen, you can change it by modifying your SQL statements or by using hints (as described in Chapter 30, “Using Hints”).

EXPLAIN PLAN can help you rewrite your SQL statements to take better advantage of such things as indexes and hash keys. By analyzing the output, you may be able to provide hints that the Oracle optimizer can use to take better advantage of your knowledge of your data. By using hints, you may be able to take better advantage of features such as the Oracle Parallel Query option.

Together, SQL Trace, EXPLAIN PLAN, and application registration can all assist in optimizing your SQL statements and your application. By using these features, you can enhance the performance of your system.

Chapter **26**

Tuning SQL Statements

Tuning your SQL statements may be one of the most important tasks you can do to improve the performance of your Oracle system. By tuning your SQL statements to be as efficient as possible, you use your system to its full potential. Some of the things you can do to improve the efficiency of your SQL statements may involve as little effort as rewriting the SQL to take advantage of some property of your database or perhaps even changing the structure of the database itself.

Tuning SQL really falls into two separate categories:

- ◆ **Tuning an existing application.** This approach involves less flexibility in terms of changing the structure of the application and the database, but may provide performance improvements anyway.
- ◆ **Designing a new application.** With a new application, you have the flexibility to design the application and perhaps even the database itself. With this approach, you can take advantage of indexes, clustering, and hashing.

This chapter looks at both of these categories. The amount of changes you can do and the flexibility you have in changing the design of the application and database depends on your particular situation. The more flexibility you have in designing the database along with the application, the better your overall results will be.

Tuning an Existing Application

Tuning an existing application can be easier in some respects and harder in others. It is easier in terms of determining the data access patterns and the specific problem areas because the application is already running and can be profiled very easily with SQL Trace and EXPLAIN PLAN. However, fixing the problems can be a challenge because you may not have the flexibility to do so.

With an existing application, you may or may not have the flexibility to fix the problem. For example, you may have system performance problems, but the system is still stable enough and performs well enough for the users to get their work done. It is hard to justify the downtime involved in reconfiguring the system when it is still in a somewhat functional state.

In this type of situation, it is important to plan far in advance and take advantage of scheduled downtime to implement system enhancements and fix any current problems and anticipated additional growth in user activity. This may be done by changing Oracle or OS parameters or by adding more disk drives, memory, and so on. If you can afford to build in an additional 20 to 30 percent growth, you may save yourself some reconfiguration time down the road.

Over the years, I have found that you take advantage of additional capacity much faster than most people anticipate and plan for. I remember the old days of the PC industry, when it was inconceivable that anyone would ever need more than 64 megabytes of memory or even think of needing a gigabyte of disk space on a desktop PC. Today, high-end PC servers support more than a gigabyte of RAM and are getting close to supporting a terabyte of disk space.

Problem Analysis

To tackle the problem of tuning an existing Oracle application, I recommend using some kind of methodology (refer to Chapter 4, “Tuning Methodology”). Here are the steps that will hopefully lead you to the problem’s resolution:

1. **Analyze the system.** Decide if there is a problem; if there is one, what is it?
2. **Determine the problem.** What do you think is causing the problem? Why?
3. **Determine a solution and set goals.** Decide what you want to try and what you think will result from your changes. What is it you want to accomplish? Do you want more throughput? faster response times? what?
4. **Test the solution.** Try it. See what happens.
5. **Analyze the results.** Did the solution meet the goals? If not, you may want to return to step 1, 2, or 3.

By following a plan of this sort, you will find it much easier to resolve the problem—or determine if there is a problem. It is possible to spend a lot of time trying to solve a performance problem that may not even exist.

By looking at the performance of the system as it is and carefully examining its characteristics, you may be able to determine whether any action is necessary to fix the problem and how much effort is involved. As mentioned in Chapter 4, I like to classify performance problems into one of three categories:

- ◆ **It's broken.** Performance is severely handicapped because of a configuration error or an incorrect parameter in the OS or RDBMS. Problems that fall into this category cause a performance swing of 50 percent or more. Problems that fall into this category are usually oversights during the system build, such as incorrectly building an index or forgetting to enable asynchronous I/O. This category of problem may indicate a more serious problem such as insufficient disk drives or memory.
- ◆ **It's not optimized.** Performance is slightly degraded because of a small miscalculation in parameters or because system capacity is slightly exceeded. These types of problems are usually easily solved by fine-tuning the configurations.
- ◆ **Not a problem.** Don't forget that sometimes there isn't a problem; you are just at the capacity of the system. This "nonproblem" is easily solved by upgrading or adding more capacity. Not all problems can be solved with tuning.

In the first case, you may have to perform some drastic fix that probably involves rebuilding the database in some fashion. The solution may be as drastic as having to rebuild from scratch to add more disk drives; it may be as simple as adding an additional index.

In the second case, you may be able to tune the system with an OS or database configuration parameter. This is usually very easy to do and does not involve much risk. However, this solution does not usually result in a huge increase in performance.

The final case may involve adding an additional CPU (if you have an SMP or MPP machine) or upgrading to new hardware. Perhaps you will find that there really isn't a problem after all and that everyone is happy with the performance of the system. One thing to remember: You rarely see the end users if everything is going fine. It's only when there are performance problems that you hear from them.

Tuning the Application

When analyzing the SQL statement, you should look for two things:

- ◆ **The SQL statement.** What does it do?
- ◆ **The effect of the SQL statement.** What is it doing it to? How does it fit into the big picture?

By looking at the SQL statement from these different angles, you may find a problem that you wouldn't find by just looking at it from one viewpoint.

For example, consider an application that does not use cached sequences to generate a primary key value. By itself, there is nothing wrong with this approach and the application is probably very efficient. But add a thousand users executing the same application and the problem is quite apparent: You have contention getting the value for the primary key value.

The SQL Statement

The best way to go about tuning the SQL statements of an existing application is to follow these few steps:

1. Familiarize yourself with the application. You should be familiar not only with the specific SQL statements but with the purpose of the application and what it does.
2. Use the SQL Trace facility to analyze what the particular SQL statements are doing, what features of the RDBMS are being used, and how well those features are being used.
3. Use `EXPLAIN PLAN` within SQL Trace to analyze how the optimizer is executing those SQL statements.

Now take a look at some specifics of these steps.

Familiarize Yourself with the Application

Look not only at the SQL statements themselves but at the effect of those statements. Make a chart of the different SQL statements and determine the number of accesses each SQL statement makes to each table in the database. This chart can give you an effective visual idea of which tables are being accessed most frequently. Consider the example shown in Figure 26.1.

This chart is a good quick reference for which SQL statements are affecting which tables. You can take this a step further and split the chart into different types of statements such as `SELECTS`, `INSERTS`, `UPDATES`, `DELETEs`, and so on. Depending on your system and whether your application is shrink-wrapped or developed in-house, this may be or may not be practical.

Figure 26.1

An example of an SQL statement-analysis chart.

SQL Statement #	Table	Act 1	Act 2	Finance 1	Finance 2	History 1
1		R W	R W	R W	R W	W
2		R W	W		R W	W
3		R W	R W		W	W
4		R		R	R	W
5		W	W		R	W

R = Read
W = Write

Use SQL Trace To Analyze the SQL Statements

By running SQL Trace on the SQL statements, you can gather much valuable information about the specific operation of each of the SQL statements. SQL Trace provides such valuable information as the following:

- ◆ Parse, execute, and fetch counts
- ◆ CPU and elapsed times
- ◆ Physical and logical reads
- ◆ Number of rows processed
- ◆ Library cache misses

You can use this information to determine which SQL statements are efficient and which ones are not. Look for the following indications of inefficient statements:

SQL Trace Output	Comments
CPU and elapsed time	If the CPU or elapsed times are very high, this SQL statement is a good candidate for tuning. It doesn't make much sense to spend time tuning statements that don't use many resources.
Executes	Focus on SQL statements that are frequently executed. Don't spend time on SQL statements that are infrequently used.

continues

<i>SQL Trace Output</i>	<i>Comments</i>
Rows Processed	An SQL statement with a high number of rows processed may not be using an index effectively.
Library Cache	A large number of library cache misses may indicate a need to tune the shared pool or to change the SQL statement to take advantage of the shared SQL area.

These clues may point you in the direction of the SQL statements that need to be tuned. You may have to alter these SQL statements to improve their efficiency. By using EXPLAIN PLAN, you may find additional areas that can be improved. For more information about SQL Trace, refer to Chapter 25, “Using EXPLAIN PLAN and SQL Trace.”

Use EXPLAIN PLAN To Analyze Statement Execution

By running EXPLAIN PLAN as part of the SQL Trace report, you can get a better idea of how the SQL statement is actually being executed by Oracle. This information (and the information supplied by SQL Trace) helps you judge the efficiency of the SQL statement. Here is a list of some of the things to look for:

- ◆ Are the table’s indexes being used when they should be? If not, the statement may not be supplying the correct parameters in the WHERE clause.
- ◆ Are indexes being used when they should not be? In cases when you are selecting too much data, you may want to use the FULL hint to bypass the index.
- ◆ What is the cost of the SQL statement? This value is given in the *position* column of the first row of the table returned by EXPLAIN PLAN.
- ◆ What is the amount of overhead incurred from SELECT or UPDATE operations?
- ◆ Is the statement being parallelized? You may have to provide a hint to effectively take advantage of the Parallel Query option.

You should ask these questions and your own specific questions as you review the EXPLAIN PLAN output. By knowing what your application is supposed to do, you may find important information about the efficiency of your statements by looking at this information. For more about the use of EXPLAIN PLAN, refer to Chapter 25.

The Effect of the SQL Statement

In addition to looking at the SQL statement itself, you should also look at the effect of the SQL statements. In many cases, some detail that is unimportant by itself can become a problem when the application and SQL statements are run by hundreds or thousands of users at the same time. The effect of this can be a bottleneck on a specific table or even a specific row. Here is a list of some things to look for when analyzing the effect of the SQL statements:

- ◆ Is the SQL statement updating a specific row? If you update a specific row as a counter, it may cause a bottleneck.
- ◆ Where is the majority of the table activity? Is a specific table being heavily accessed? This could indicate an I/O bottleneck.
- ◆ Is there significant `INSERT` activity? Is it all to one table? This may indicate a contention problem on a certain table.
- ◆ How much activity is there? Can the system handle it? You may find that the SQL statements overload your particular system.

These are just a few of the things to consider when you are looking at the effects of the application on the system. I have seen cases in which an application, fully tested in the lab, moves into production and fails because it was tested with only one or two users. It is important to take into account the effect of hundreds or thousands of users simultaneously accessing the application.

Review of How To Tune an Existing Application

Tuning an existing application can be quite a challenge. Determining whether the system is in need of optimization—and figuring out how to do it—is not always easy. The task may be easier if you take a methodical approach like this one:

1. **Analyze the situation.** It may be that your system is not in need of adjustment. I do not recommend making any changes to a stable system unless you have to.
2. **Familiarize yourself with the application.** Look at the SQL statements as well as the overall application. Understand the purpose of the application.
3. **Make an analysis chart.** Look at the table accesses being generated by the application.
4. **Run SQL Trace with EXPLAIN PLAN.** See what the SQL statements are really doing. Choose the statements to focus on based on how often they are used and how many resources they use.
5. **Understand how these SQL statements affect the server system.** Look at Oracle and the OS. Determine which disks may be overused and where contention could occur when many users run the application.

With an existing application, you may or may not have the flexibility to fix the problem. I do not recommend making any changes to an existing application or a functioning system unless some specific performance problems are affecting users or limiting the capacity of the system.

Of course, if you have the flexibility to make changes and there is a need, any of the design and application changes described in the following section, “Designing a New Application,” also apply to an existing application.

Designing a New Application

Although this part of the chapter is directed at tuning SQL statements associated with new applications, it may be appropriate for you to make these changes to existing applications—if you have the flexibility to do so. The reason these guidelines are separate is because many of them involve not only tuning the SQL statements and application, but changing the database schema as well.

In the design stage, it is important to plan the application and the database design together. By properly designing the application to take advantage of the design and features of the database, you can take optimal advantage of both of them. At the same time, the database should be designed to function properly with the application that uses it. The design of the database should reflect the purpose of the application. The following sections look at some of the optimizations that are possible.

Indexes

An index is an optional structure designed to help you achieve faster access to your data. Just like the index in this book, an Oracle index is logically and physically independent of the data in the associated table or cluster. You can use the index to speed access to the data, or you can retrieve the data independently from the index by searching the tables for it. When optimally configured and used, indexes can significantly reduce I/O to the data files and greatly improve performance.

The presence of an index is transparent to the user or application and requires no application changes. However, if you know of the existence of the index and design the application to take advantage of the index, you can greatly reduce the I/Os necessary to retrieve the desired data. The only indication of an index may be improved time to access the data.

Once an index has been created on a table, the maintenance of that index is done automatically by Oracle. Inserts, updates, and deletions of rows are automatically updated in the related indexes.

A table can have any number of indexes, but the more indexes there are, the more overhead is incurred during table updates, inserts, and deletions. This overhead is incurred because all associated indexes must be updated whenever table data is altered.

TIP: If you use the Oracle Parallel Query option, you can create the index in parallel, thus reducing the time it takes to create the index.

Index Types

There are several different types of indexes:

<i>Index Type</i>	<i>Description</i>
Unique index	A unique index is an index value that has the additional constraint that it cannot be duplicated. Although this constraint may be specified, it is usually better to associate this constraint with the table itself rather than with the index. Oracle enforces unique integrity constraints by automatically defining a unique index on the unique key.
Nonunique index	A nonunique index does not impose the constraint that the index value be unique. Such an index is useful if you need quick access to a nonunique value.
Cluster index	A cluster index is an index created on the cluster key in an Oracle cluster. It is required that a cluster key be indexed.
Composite index	A composite index is an index on several columns in a table. The column values can be in any order and the columns do not have to be adjacent in the table.

A composite index is useful when `SELECT` statements have `WHERE` clauses that reference several values in the table. Because the index is accessed based on the order of the columns used in the definition, it is wise to base the index order on the frequency of use. The most referenced column should be defined first, and so on.

The index should be created based on the values accessed in the application; the application should be developed to take advantage of the indexes. Having knowledge of and influence over these indexes can be very useful to the application developer.

What To Index

The index is usually determined by the column values that are indexed. Remember that the more indexes on a table, the more overhead is incurred during updates, inserts, and deletes. It is important to index selectively.

Which Tables Should Be Indexed?

Use the following guidelines to decide which tables to index:

- ◆ Index tables when queries select only a small number of rows. Queries that select a large number of rows defeat the purpose of the index. Index the table when queries access less than 5 percent of the rows in the table.

- ◆ Don't index tables that are frequently updated. Update, insert, and deletes on indexed tables incur extra overhead. Base your decision about whether or not to index a table on the number of updates, inserts, and deletes relative to the total number of queries to the table.
- ◆ Index tables that don't have duplicate values on the columns usually selected in WHERE clauses. Tables for which the selection is based on TRUE/FALSE values are not good candidates for an index.
- ◆ Index tables that are queried with relatively simple WHERE clauses. Complex WHERE clauses may not be able to take advantage of indexes. You can solve this shortcoming by creating a complex index, by simplifying the SQL statement, or by using a hint.

Once you decide to use an index, you must then decide which columns to put the index on. You may index one or more columns, depending on the table.

Which Columns Should Be Indexed?

Use the following guidelines to decide which columns to index:

- ◆ Choose columns frequently specified in WHERE clauses. Frequently accessed columns can most benefit from indexes.
- ◆ Don't index columns that do not have many unique values. Columns in which a good percentage of rows are duplicates cannot take advantage of indexing.
- ◆ Columns that have unique values are excellent candidates for indexing. Oracle automatically indexes columns that are unique or that are primary keys defined with constraints. These columns are most effectively optimized by indexes.
- ◆ Index columns that are foreign keys of referential integrity constraints in cases where large numbers of concurrent INSERT, UPDATE, and DELETE statements access both the parent and child tables. Such an index allows the child table to be updated without having to lock the parent table.
- ◆ Columns that are commonly used to join tables are good candidates for indexing.
- ◆ Frequently modified columns probably should not be index columns because of the overhead involved in updating the index.

NOTE: Remember that some penalty is associated with performing INSERT, UPDATE, and DELETE statements on columns that are indexed. If you have a high number of those statements, an index may hurt more than help. Use SQL Trace on the SQL statements that access that table both with and without an index. Compare the results.

Composite Indexes

Composite indexes may be more effective than individual indexes in situations such as the following:

- ◆ When two columns are not unique individually but are unique together, composite indexes may work very well. For example, columns A and B have few unique values, but rows with a particular A AND B are mostly unique. Look for WHERE clauses with AND operators.
- ◆ If all values of a SELECT statement are in a composite index, the table is not queried; the result is returned from the index.
- ◆ If several different queries select the same rows by using different WHERE clauses based on different columns, consider creating a composite index with *all* those columns used in the WHERE statements.

If carefully designed, composite indexes can be quite useful. As with single-column indexes, they are most effective if applications are written with the indexes in mind.

Once you create the index, periodically use SQL Trace to determine whether your queries are taking advantage of the index. It may be worth the effort to try the query with and without indexes and compare the results to see whether the index is worth the space it is using.

How To Avoid an Index

If you have an application that can take advantage of an index, but contains a few SQL statements that result in poor performance when they use an index, you can tell the optimizer to bypass the index. There are several ways this can be done:

- ◆ Write the SQL statement to avoid using a SELECT statement on an indexed column. By not selecting the column or set of columns that are indexed, you avoid the index.
- ◆ Use hints. When you use hints in your SQL statements, you can tell the optimizer not to use the index for this particular SQL statement. Hints are detailed in Chapter 30, “Using Hints.”

In this manner, you can design your database to effectively use indexes and have the flexibility to avoid the index when it is not optimal to do so.

Review of Indexes

If used properly, indexes can significantly improve performance in your system. You must first decide whether an index is appropriate for the data and access patterns in your particular system. Having decided to use an index, you must decide which columns to index.

Indexing an inappropriate column or table can actually reduce performance. Indexing appropriately can greatly improve performance by reducing I/O and speeding access times. If necessary, use indexes and take advantage of hints to avoid the index when it would be inefficient to use it.

Careful planning and periodic testing with SQL Trace can lead to a very effective use of indexes, with optimal performance as the outcome.

Clusters

A *cluster*, sometimes called an *index cluster*, is an optional method of storing tables in an Oracle database. In a cluster, multiple related tables are stored to more efficiently improve access time to the related items. Clusters provide the most benefit when the related data is frequently accessed together. The existence of a cluster is transparent to users and to applications; the cluster only affects how data is stored.

Clusters can be advantageous in certain situations and disadvantageous in others. You must be careful in determining whether a cluster can help performance in your configuration. Typically, clusters are advantageous if the clustered, related data is used primarily in joins because the data to be used in the join is retrieved together in one I/O operation.

If you have two tables with related data that are frequently accessed together, having a cluster can improve performance by preloading the related data into the SGA. Because you frequently use the data together, having that data already in the SGA greatly reduces access time.

If you do not typically use the information together, you will find no performance benefit from using a cluster. There is even a slight disadvantage because the SGA space is taken up by the additional table information.

Another disadvantage of clusters is a reduction in the performance of `INSERT` statements. This happens because of the additional complexity of the use of space and because there are multiple tables in the same block. The clustered table also spans more blocks than the individual table would, causing more data to be scanned.

Review of Clusters

A cluster may or may not be useful, depending on how the data is primarily accessed. To decide whether you can benefit from a cluster, consider these guidelines:

- ◆ Cluster tables where data is primarily accessed together in a join. The reduced I/O required to bring the additional data into the SGA and the fact that the data is already cached can be a big advantage.
- ◆ Do not cluster tables that see a large number of `INSERT` statements.
- ◆ Do not cluster tables if the data in those tables is not frequently accessed together.
- ◆ Do not cluster tables if full-table scans are often performed on only one of the tables in the cluster. The additional space required by the cluster and additional I/O reduce performance.

By following these guidelines, you should be able to determine whether a cluster is right for your installation. Be careful: a cluster can hurt performance if improperly used.

Hash Clusters

A *hash cluster* is similar to a cluster but uses a hash function rather than an index to reference the cluster key. A hash cluster stores the data based on the result of a *hash function* (a numeric function that determines the data block in the cluster based on the value of the cluster key).

To find the data block in an index cluster, there must first be one or more I/Os to the cluster index to find the correct data block. In a hash cluster, the cluster key itself tells Oracle where the data block is. This arrangement can reduce to one the number of I/Os required to retrieve the row.

In contrast to the index cluster, which stores related data together based on the row's cluster key value, the hash cluster stores related rows together based on their hash values.

The number of hash values is determined by the value of the `HASHKEYS` parameter of the `CREATE CLUSTER` command. The number and size of the cluster keys is very important and should be carefully calculated.

When To Use Hash Clusters

The decision to use hash clusters is an important one. Hash clusters can be beneficial because, when they are effectively used, the requested data can be retrieved in just one I/O. Unfortunately, if a hash cluster is used on a table that is not a good candidate for hashing, performance can be severely degraded.

Although hash clusters can be used in a similar fashion to index clusters, you do not have to cluster the tables to use a hash cluster. In fact, in many cases, it is useful to create a single table as a hash cluster. By using hashing, you can retrieve your data with a single I/O rather than the multiple I/Os required to retrieve the same data using a B -Tree index.

Because hashing uses the value of the data to calculate the data block the desired data is in, hashing is best used on tables that have unique values for the cluster key and that are queried primarily by equality queries on the cluster key. In the case of equality queries, the data is usually retrieved in one read operation. The cluster key need not be a single column; if the typical query uses an equality on a set of columns, use these columns to create a composite key.

An good candidate for hashing has the following properties:

- ◆ **Unique cluster keys.** Hashing performs best when the value of the cluster key is fairly unique. Because the data is laid out based on the key value, a column with a lot of duplicates is not a good candidate for the hash key.
- ◆ **Equality queries.** The majority of queries are equality queries on the cluster key. This type of query reaps the greatest benefit from the hash cluster.
- ◆ **Static size.** Hashing is optimal when the table or tables are fairly static in size. If the table stays within its initial storage allocation, you do not see any performance degradation from using a hash cluster; but if the table grows out of its initial allocation, performance can degrade. In this case, overflow blocks are required.

- ◆ **Constant cluster key.** Hashing can also degrade performance when the value of the cluster key changes. Because the location of the block in which the data resides is based on the cluster key value, a change in that value can cause the row to migrate to maintain the cluster.
- ◆ **No table scans.** Hashing can degrade the performance of table scans because the scan must read blocks that may not have much data in them. Because the table is originally created by laying out the data in the cluster based on the value of the cluster key, there may be some blocks that have few rows.

Do not use a hash cluster on a table if the application frequently modifies the cluster key or the table is constantly being modified. Because the cluster key is based on a calculation, you can incur significant overhead by constantly recalculating the key.

Any time you have a somewhat static table with a unique column value or set of column values, consider creating a hash cluster.

Review of Hash Clusters

As with index clusters, hash clusters have both advantages and disadvantages. Hash clusters are very efficient in retrieving data based on equality queries on the hash key. If you do not retrieve data based on that key, the query is not hashed. As with the index cluster, you see a performance decrease when executing `INSERT` statements on a hash cluster.

With both index clusters and hash clusters, carefully consider the access patterns to the tables before deciding whether a cluster can help performance. A wrong decision can end up costing you in terms of performance.

If you can take advantage of hashing by meeting somewhat strict criteria, you will see very good performance. Hashing is extremely efficient under the right conditions.

Packages, Procedures, and Functions

Another way to improve performance of your SQL statements is by using packages, procedures, and functions. *Packages* can help improve performance by storing together procedures and functions that are often used together. By storing these elements together, you can reduce the I/O required to bring them into memory from disk. Because these elements are often used together, they can also be loaded from disk together.

By using *stored procedures*, you benefit in several ways. Stored procedures allow you to reduce the amount of data sent across the network. The stored procedure requires fewer instructions to be sent to the server; in many cases, less data must be sent back to the client from the server.

A second benefit of a stored procedure is the increased chance that the SQL statement can be used by other processes. Because the SQL statement is defined and used by many processes, chances are good that the SQL statement will already be parsed in the shared SQL area and available to other users.

Chapter 28, “Using Procedures, Functions, and Packages,” covers how to use these elements to improve the performance of SQL statements.

Optimization Approaches

Oracle offers several options for optimization techniques. Among these are a cost-based approach and a rule-based approach. The approach you take depends both on your application and your data. In most cases, the cost-based approach is recommended because it determines an execution plan that is as good or better than the rule-based approach.

The following sections look at the optimization approaches available from Oracle and when each approach is appropriate. Remember that you can use hints to specifically tell Oracle how you want the SQL statement to be executed. There are several ways you can indicate your preference, as described later in this chapter.

The discussion here is limited to an overview because the Oracle optimizer is detailed in Chapter 27, “Using the Oracle Optimizer.”

Rule-Based Approach

The rule-based approach to Oracle optimization is straightforward and consistent. In the rule-based approach, the execution plan is derived by examining the available paths and ranking them against a list of predetermined values for these paths (see Figure 26.2).

Figure 26.2

The rule-based optimization rankings.

Rank	Access Path
1	Single row by ROWID
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite key
9	Single-column indexes
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort merge join
13	MAX or MIN on indexed columns
14	ORDER BY on indexed columns
15	Full-table scan

With the rule-based optimization approach, the optimizer determines the ways to execute the SQL statement. If there is more than one way to execute the SQL statement, the table in Figure 26.2 is used to choose the approach with the lowest ranking.

Cost-Based Approach

The cost-based approach to optimization uses existing knowledge of the database to choose the most efficient execution plan. During the normal operation of the RDBMS, statistics are gathered on the data distribution and storage characteristics. The optimizer uses this information to determine the most optimal execution plan.

This optimization approach takes three steps:

1. The optimizer generates a set of possible execution plans, just as it does with the rule-based optimization approach.
2. The cost of each plan is determined based on statistics gathered about the database. This cost is based on CPU time and the I/O and memory necessary to execute the plan.
3. The optimizer compares the costs and chooses the execution plan with the lowest cost.

The cost-based approach is usually preferred. In some cases, the rule-based approach may be more appropriate, as discussed in Chapter 27, “Using the Oracle Optimizer.”

Hints

You can use hints to inform the optimizer of some special facts you know about the data or the SQL statement that may affect the execution plan. By using hints, you indicate that the SQL statement may be more efficient by using a certain execution plan (for example, a full-table scan or increased parallelism). Hints are detailed in Chapter 30, “Using Hints.”

Review of Optimization Approaches

Oracle offers different optimization approaches. Among these are a cost-based approach and a rule-based approach. The approach you take depends both on your application and your data. In general, the cost-based approach is the recommended approach. In most cases, the cost-based approach determines an execution plan that is as good or better than the rule-based approach.

The optimization approaches and the use of hints are discussed in later chapters. By properly optimizing your system, you can achieve the most efficient execution of your SQL statements.

Discrete Transactions

Discrete transactions are an optional way of processing SQL statements. They may help performance in certain situations. Discrete transactions can be used only in a limited set of circumstances, but if you can take advantage of them, they can help performance.

Discrete transactions work with small, nondistributed transactions and improve performance by deferring the writing of redo information until the transaction has been committed. Discrete transactions cannot—and should not—be used for all transactions.

How Do Discrete Transactions Work?

With discrete transactions, all changes made to data are deferred until the transaction has been committed. Even though redo information is saved, it is not written to the redo log until the transaction has been committed. Until the commit, the redo information is stored in another area of memory.

When the transaction is committed, the redo information is written to the redo log and the changes are made to the data block. This arrangement causes the rollback segments to be bypassed. Because the changes are deferred until the commit, the undo information does not have to be saved.

With normal transactions, the undo information must be saved in the rollback segments because the data blocks have already been changed. With discrete transactions, because the data blocks have not been changed until the commit, it is unnecessary to save that information. Discrete transactions should not be used on data accessed with long-running queries because those queries will not be able to access any undo information.

When Should Discrete Transactions Be Used?

Discrete transactions should be used only under certain circumstances and with certain transaction types. Here are some guidelines for when to use discrete transactions:

- ◆ Discrete transactions should be used to modify only a few data blocks. The amount of data and the amount of time it takes to perform the transactions should be as small as possible.
- ◆ Discrete transactions should not be used on data that might be accessed by long-running queries. Those queries will not be able to access any undo information.
- ◆ Discrete transactions should not be used on any rows containing LONG data.

Because of these restrictions, discrete transactions can be used only with a small subset of transactions. These transactions should be small and quick. If you can take advantage of discrete transactions, you should see a fairly good performance benefit.

Review of Discrete Transactions

Discrete transactions work with small, nondistributed transactions and can improve performance by deferring the writing of redo information until the transaction has been committed. Discrete transactions cannot and should not be used for all transactions.

Before making modifications to your application design, determine whether discrete transactions can benefit your application. Determine whether your transactions fit the profile that can benefit from discrete transactions; also determine whether these changes will be significant enough to warrant the use of discrete transactions.

If your system is under a heavy load of mostly small transactions, it may be worth investigating discrete transactions. Try implementing them and see what kind of benefit they return.

Summary

Tuning your SQL statements is one of the most important tasks you can do to improve performance. In fact, you should tune your SQL statements before tuning your RDBMS server. By tuning your SQL statements to be as efficient as possible, you can use your system to its full potential. Some of the things you can do to improve the efficiency of your SQL statements may involve as little effort as rewriting the SQL to take advantage of a property of your database or may involve changing the structure of the database itself.

As you have seen, tuning SQL falls into two categories:

- ◆ Tuning an existing application
- ◆ Designing a new application

By designing your system with the goals of optimization and performance, you can take advantage of numerous Oracle features. By far, the best-tuned systems are those tuned from the design stage. This chapter described many ways you can tune the new application; if you have enough flexibility, you can make these changes to an existing application as well.

Chapter 27

Using the Oracle Optimizer

During the execution of SQL statements, Oracle chooses an execution plan by which these statements are executed. The execution plan is determined by the Oracle optimizer by using the optimization approach specified in the initialization parameters. The optimization approach can be overridden by the use of hints, as detailed in Chapter 30, “Using Hints.”

The effectiveness of the execution plan depends mainly on the optimization method chosen. This optimization method can consist of either a rule-based or cost-based approach. Which approach you take depends on both your application and your data.

How the Optimizer Works

To understand how the optimizer optimizes your SQL statements, it is useful to first look at how the optimizer works. When an SQL statement is parsed and passed off to the optimizer, the following occurs:

1. The SQL statement is analyzed and evaluated. In this first stage, the SQL statement is essentially checked by the optimizer.
2. The SQL statement is modified by the optimizer. If the statement is complex, the optimizer may change the statement to be more effectively processed, if necessary.
3. View merging is performed. If the statement is accessing a view, the Oracle optimizer sometimes merges the query in the statement with a query in the view before optimization.
4. The choice of optimization method is made. The optimizer chooses between a cost-based and rule-based approach, based on the amount of analysis data available for the object and also on any hints.
5. The access path is chosen. The optimizer chooses one or more access paths to each table referenced in the SQL statement.
6. The join order is chosen. If more than one join is done in the SQL statement, Oracle chooses the most appropriate order in which the joins will occur.
7. The join operations are chosen. The optimizer chooses the most appropriate operations to use to perform the joins.

As you can see, the process through which the optimizer chooses the most optimal execution plan for the SQL statement is quite complex. But it is in this fashion that the best execution plan is usually chosen.

I could go into detail here about how the Oracle optimizer expands statements at the lowest level, but that is getting out of the scope of this book. It is enough to know that the optimizer breaks the statements into their constituent components and determines their individual costs.

How To Specify an Optimization Mode

The optimization mode (cost-based or rule-based) can be chosen using either of the following methods:

- ◆ The `OPTIMIZATION_MODE` initialization parameter
- ◆ The `OPTIMIZER_GOAL` parameter of the `ALTER SESSION` command

When specifying the optimization method with either the initialization file or the `ALTER SESSION` command, you can specify the following options:

<i>Option</i>	<i>Description</i>
CHOOSE	This option allows the Oracle optimizer to choose an optimization mode based on the availability of statistics on a particular table, cluster, or index. If statistics are available for any of the tables accessed in the SQL statement, the cost-based approach is used with the goal of best throughput. If none of the tables has statistics available, the rule-based approach is used; the rule-based approach is also the default if no optimization approach is specified.
RULE	This option causes the Oracle optimizer to always use the rule-based optimization approach, regardless of any statistics that may have been gathered for the tables being accessed.
ALL_ROWS	This option causes the optimizer to use the cost-based approach on all SQL statements, even if there are no statistics available for the tables being accessed. This approach has the goal of best throughput, with the least amount of system resources being used.
FIRST_ROWS	This option causes the optimizer to use the cost-based approach on all SQL statements, even if no statistics are available for the tables being accessed. This approach has the goal of best response time.

NOTE: If the cost-based optimization approach is used and no internal statistics are available for a table that is being accessed, other information (such as the number of data blocks in the table) is used to estimate the cost of various operations.

Optimization Methods

In general, the cost-based approach is the recommended approach. In most cases, the cost-based approach determines an execution plan that is as good or better than the rule-based approach. However, if you have manually tuned your SQL statements, you may get better performance with rule-based optimization than cost-based optimization.

The rule-based approach can be useful if you are moving a highly tuned application from an older version of Oracle that has been using the rule-based approach. This and a lack of statistics may cause rule-based optimization to be more efficient than the cost-based approach. However, as you gather statistics on your database, you may want to migrate to cost-based optimization.

The following sections examine both approaches. Hints can also be very useful in optimizing the execution of your SQL statements; hints are discussed in Chapter 30, “Using Hints.”

Rule-Based Approach

The rule-based approach to Oracle optimization is the simpler of the two methods. In the rule-based approach, the execution plan is derived by examining the available paths and comparing them against a table of the rank of these paths. The table of costs is shown in Table 27.1.

Table 27.1 Cost of Access Paths for Rule-Based Optimization

Rank	Access Path
1	Single row by ROWID
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite key
9	Single-column indexes
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort-merge join
13	MAX or MIN of indexed column
14	ORDER BY on indexed columns
15	Full-table scan

Because the rule-based approach is based simply on the SQL statements themselves, it is unnecessary to have any statistics about the database tables. The rule-based approach follows these steps to determine the execution plan:

1. Determine the possible execution plans.
2. Rank the different plans according to Table 27.1.
3. Choose the approach with the lowest ranking.

In this way, the rule-based optimization approach is very efficient and works well. However, if statistics are available for your tables, clusters, or indexes, the cost-based approach can be very efficient.

Cost-Based Approach

The cost-based approach to optimization uses information about your database to choose the most efficient execution plan. During the normal operation of the RDBMS, or when you execute the `ANALYZE` command, statistics are gathered on the data distribution and storage characteristics for your database tables, clusters, and indexes. The cost-based optimizer uses this information to determine the most optimal execution plan.

This approach is done in three steps:

1. The optimizer generates a set of possible execution plans, just as the rule-based optimization approach does.
2. The cost of each plan is determined based on statistics gathered about the database. This cost is based on CPU time, I/O, and memory necessary to execute the plan.
3. The optimizer compares the costs and chooses the execution plan with the smallest cost based on your specifications.

The default goal of the cost-based optimizer is to generate an execution plan that gives the best throughput. You can specify other optimization goals, including the following:

<i>Optimization Goal</i>	<i>Description</i>
Minimal Resources	This goal causes the optimizer to choose the execution plan that uses the least amount of system resources.
Best Response Time	This goal causes the optimizer to choose the execution plan that has the best response time.

By choosing the optimization approach that best suits your particular installation and application, the performance of your SQL statements can be tuned to specifically meet your needs.

Using the ANALYZE Command

You can use the `ANALYZE` command to gather statistics about your system that can be used for the cost-based optimizer. This command can be used not only for statistics gathering but for other purposes as well. The `ANALYZE` command can be used to do the following:

<i>Function</i>	<i>Description</i>
Gather statistics	The <code>ANALYZE</code> command can be used to gather statistics about tables, clusters, and indexes that can assist the cost-based optimizer in choosing the best execution plan for your system.

continues

<i>Function</i>	<i>Description</i>
Check data integrity	The <code>ANALYZE</code> command can be used to validate the integrity of the structure of a table, index, or cluster.
Chained-row statistics	The <code>ANALYZE</code> command can be used to gather statistics about the number of chained rows in a table or cluster.

The statistics gathered by the `ANALYZE` command can better help the optimizer make the correct choice in determining an execution plan.

How To Run the `ANALYZE` Command

How you run the `ANALYZE` command is determined by the type of statistics or analysis you want to perform. The `ANALYZE` command can be used in several different modes. The mode you choose depends on the data you want to gather as well as on the configuration of your system.

Using `ANALYZE` To Gather Statistics

You can use the `ANALYZE` command to gather statistics in one of two modes. The first mode scans the entire table, cluster, or index and calculates statistics exactly, based on your data. Although this is the most accurate method, it requires enough temporary space to hold and sort all the rows of the table or cluster (no space is required for an index). Computing the statistics also uses a great deal of system resources.

The second mode of the `ANALYZE` command estimates statistics. This method performs a sampling of the table, cluster, or index in order to estimate statistics. In this method, the entire table or cluster is not scanned; a portion of the data is used to determine the statistics. The amount of data used can be specified when you invoke the `ANALYZE` command.

Using `ANALYZE` To Compute Exact Statistics

To use the `ANALYZE` command to compute exact statistics, invoke `ANALYZE` with one of the following syntaxes.

For Tables:

```
ANALYZE TABLE table_name
    COMPUTE STATISTICS;
```

For Clusters:

```
ANALYZE CLUSTER cluster_name
    COMPUTE STATISTICS;
```

For Indexes:

```
ANALYZE INDEX index_name
    COMPUTE STATISTICS;
```

Using ANALYZE with the COMPUTE STATISTICS option scans the entire table, cluster, or index and computes exact statistics. When you compute the exact statistics, the resultant data is more accurate than that achieved by estimating the statistics; however, you use much more system resources to get this information. When you compute statistics for tables and clusters, you must have enough temporary space to load and sort the entire table or cluster. You do not need this temporary space for indexes.

Using ANALYZE To Estimate Statistics

When you use the ANALYZE command to estimate statistics, Oracle does much less work and uses much less temporary space. To run the ANALYZE command to estimate statistics, use one of the following syntaxes.

For Tables:

```
ANALYZE TABLE table_name  
ESTIMATE STATISTICS;
```

For Clusters:

```
ANALYZE CLUSTER cluster_name  
ESTIMATE STATISTICS;
```

For Indexes:

```
ANALYZE INDEX index_name  
ESTIMATE STATISTICS;
```

When you use ANALYZE with the ESTIMATE STATISTICS option, Oracle scans a portion of the table, cluster, or index and computes estimated statistics. You can specify the amount of data scanned and used for statistics by including one or both of these additional parameters:

```
SAMPLE xxxx ROWS;  
SAMPLE yy PERCENT;
```

Place the SAMPLE xxxx ROWS parameter at the end of the ANALYZE command, as follows:

```
ANALYZE TABLE table_name  
ESTIMATE STATISTICS  
SAMPLE 10000 ROWS;
```

Place the SAMPLE yy PERCENT parameter at the end of the ANALYZE command, as follows:

```
ANALYZE TABLE table_name  
ESTIMATE STATISTICS  
SAMPLE 40 PERCENT;
```

Although estimating statistics does not give you as accurate a representation as computing statistics does, the lesser amount of resources consumed usually makes estimating statistics a better choice. By making the percentage of data scanned as large as possible for your system, you can increase the effectiveness of the statistics you gather.

Using ANALYZE To Check Structural Integrity

In addition to gather statistics, you can use the ANALYZE command to validate the structure of a table, cluster, or index. You should run this command only if you feel that there is some problem with the structure of these objects. These problems can occur as the result of a hardware or software problem that caused data corruption. By analyzing the structure of the schema objects, you can find any problems immediately and avoid a system crash. The ANALYZE command can be used in this manner with one of the following syntaxes:

For Tables:

```
ANALYZE TABLE table_name
  VALIDATE STRUCTURE;
```

For Clusters:

```
ANALYZE CLUSTER cluster_name
  VALIDATE STRUCTURE;
```

For Indexes:

```
ANALYZE INDEX index_name
  VALIDATE STRUCTURE;
```

Adding the CASCADE option to the ANALYZE command results in the structure of all related tables being analyzed as well. Use the following syntax:

```
ANALYZE TABLE table_name
  VALIDATE STRUCTURE CASCADE;
```

When you analyze the integrity of the structure of tables, clusters, or indexes, the command returns any structural problems. If there are problems with structure of these objects, you should drop the object, re-create it, and reload the data.

Using ANALYZE To Determine Chained Rows

You can also use the ANALYZE command to determine the extent and existence of chained or migrated rows in your table or cluster. The existence of chained or migrated rows (as described in Chapter 10, “Performance Enhancements”), if significant, can cause severe performance degradation and should be corrected. Use the ANALYZE command in this manner with one of the following syntaxes:

For Tables:

```
ANALYZE TABLE table_name
  LIST CHAINED ROWS INTO chained_rows;
```

For Clusters:

```
ANALYZE CLUSTER cluster_name
  LIST CHAINED ROWS INTO chained_rows;
```

The *chained_rows* table is a table with the proper structure to hold the information returned from the ANALYZE command. You can easily create the *chained_rows* table by using the UTLCHAIN.SQL script distributed with Oracle.

Summary of the ANALYZE Command

As you have seen, the ANALYZE command can be quite useful for gathering statistics as well as for analyzing the structural integrity of tables, clusters, and indexes. The ANALYZE command can also be used to determine the existence and extent of chained and migrated rows. By using the ANALYZE command to gather statistics, the effectiveness of the cost-based optimizer can be increased; therefore, performance itself can be increased.

Data Dictionary Statistics

When you use the ANALYZE command to create statistics for the cost-based optimizer to use, these statistics are inserted into some internal Oracle performance tables. These tables can be queried through several views. Although these views provide essentially the same information, depending on the particular view you choose, the scope of the information changes slightly. The following views are prefixed with the following characters:

<i>View</i>	<i>Description</i>
USER_	This view contains information about the objects owned by the user.
ALL_	This view contains information about the objects accessible by the user. These are objects owned by the user as well as objects with PUBLIC access.
DBA_	This view contains information on all objects in the system.

These views provide information about different parts of the system, such as tables, clusters, indexes, and columns. Following is a brief list of the views available that contain performance information:

- ◆ **Table views:** USER_TABLES, ALL_TABLES, DBA_TABLES
- ◆ **Cluster views:** USER_CLUSTERS, ALL_CLUSTERS, DBA_CLUSTERS
- ◆ **Index views:** USER_INDEXES, ALL_INDEXES, DBA_INDEXES
- ◆ **Column views:** USER_TAB_COLUMNS, ALL_TAB_COLUMNS, DBA_TAB_COLUMNS

The information contained in the internal tables referenced by these views is used by the optimizer to make decisions about which execution plan to take. The decision about which execution plan to take is also based on information about the size of the object and the data contained in the object. Some of the information contained in these tables is presented in Tables 27.2, 27.3, 27.4, and 27.5.

Table 27.2 Data for Tables in USER_TABLES, ALL_TABLES, DBA_TABLES

<i>Column</i>	<i>Description of Contents</i>
AVG_SPACE	The average amount of free space in the table.
AVG_ROW_LEN	The average length of a row.
BLOCKS	The number of blocks in the table.
CHAIN_CNT	The number of chained rows.
EMPTY_BLOCKS	The number of blocks that have never been used.
NUM_ROWS	The number of rows in the table.

Table 27.3 Data for Clusters in USER_CLUSTERS, ALL_CLUSTERS, DBA_CLUSTERS

<i>Column</i>	<i>Description of Contents</i>
AVG_BLOCKS_PER_KEY	The average number of blocks that have rows that use the same key.
CLUSTER_TYPE	The type of cluster: whether it is an index cluster or a hash cluster.
HASHKEYS	The number of hash keys if it is a hash cluster.

Table 27.4 Data for Indexes in USER_INDEXES, ALL_INDEXES, DBA_INDEXES

<i>Column</i>	<i>Description of Contents</i>
AVG_LEAF_BLOCKS_PER_KEY	The average number of leaf blocks (lowest level index blocks) per key.
AVG_DATA_BLOCKS_PER_KEY	The average number of data blocks per key.
BLEVEL	The level of the B*-Tree.
CLUSTERING_FACTOR	The amount of order or disorder in the table the index is referencing.
DISTINCT_KEYS	The number of distinct keys in the index.
LEAF_BLOCKS	The number of leaf blocks in the index.
UNIQUENESS	States whether the index is unique or nonunique.

Table 27.5 Column Data in USER_TAB_COLUMNS, ALL_TAB_COLUMNS, and DBA_TAB_COLUMNS

<i>Column</i>	<i>Description of Contents</i>
DENSITY	The density of the column (rows per data block).
HIGH_VALUE	The second-highest value in this column of the table.
LOW_VALUE	The second-lowest value in this column of the table.
NUM_DISTINCT	The number of distinct values in this column of the table.

With this information, the optimizer can more precisely determine the optimal execution path based on data about your specific system. When you run the `ANALYZE` command with the `ESTIMATE STATISTICS` option, many of these values are estimates rather than actual computed results.

Hints

You can use hints to inform the optimizer of any special facts you know about the data or the SQL statements that may affect the execution plan. By using hints, you indicate that the SQL statement may be more efficient by using a certain execution plan (such as a full-table scan or increased parallelism). Hints are detailed in Chapter 30, “Using Hints.”

Summary

When SQL statements are executed, the Oracle optimizer determines the execution plan based on the available data. The Oracle optimizer uses the optimization approach specified in the initialization parameters to determine the execution plan, as you have seen in this chapter.

The effectiveness of the execution plan depends on the optimization method chosen and the availability of good statistics for your database. When using the cost-based optimization approach, the effectiveness of the optimization can be enhanced by including more and better database performance statistics. Regardless of your system configuration, you can override or enhance the optimization approach by using hints, as discussed in Chapter 30.

This chapter described the various optimization approaches. I recommend the use of the `CHOOSE` hint under most conditions. When you specify `CHOOSE`, Oracle takes advantage of the cost-based optimizer under most situations but uses the rule-based approach when no statistics are available. In Chapter 30, you see how SQL statements can be further enhanced by using hints and information you know about your data and your database layout.

Chapter 28

Using Procedures, Functions, and Packages

Using functions, procedures, and packages can improve performance in several ways. These performance enhancements take the form of reduction in the amount of data that must be transmitted across the network and an increase in hits in the shared SQL cache.

Procedures and *functions* are subprograms made up of PL/SQL code that take a set of parameters given to them by the calling program and perform a set of actions. The only real difference between a procedure and a function is that a function can include a return value. Both functions and procedures can modify and return data passed to them as a parameter. Usually, procedures are used unless only one return value is needed.

Packages are sets of related procedures or functions compiled and stored together in the data dictionary. Packages allow you to group PL/SQL types, objects, and subprograms into a logical unit. If you link these logically related entities together, it can be easier to program and modify modules based on their function and relation. You enhance performance because the entire package is loaded into memory when it is first called, increasing the chance for a cache hit on a related function or object that is likely to be called soon.

Because a procedure, function, or package is stored within the library cache, it is available for immediate use by your applications. Because these objects are stored in an already-parsed form, performance is also improved.

Procedures, functions, and packages are used to call certain SQL statements that are used over and over again. Any set of SQL statements that you use frequently in your application can benefit from being made into a stored procedure or function.

This chapter reviews some information about the library cache and then describes how procedures, functions, and packages can be used to help improve overall performance.

Review of the Library Cache

As you know, the library cache contains the shared SQL and PL/SQL areas. By increasing the cache-hit rate in the library cache, you increase performance. This increase comes both from reducing the overhead needed to parse the SQL statements in the shared SQL area and from retrieving those statements from cache (reducing the need to retrieve those statements from disk).

A cache miss in the shared SQL area occurs either when a parse statement is called and the already-parsed statement does not exist in the shared SQL area or when an application tries to execute an SQL statement and the shared SQL area containing the parsed statement has been deallocated from the library cache.

Here is a review of the requirements necessary for an SQL statement to take advantage of the library cache. For an SQL statement to take advantage of SQL or PL/SQL statements that have already been parsed, the following criteria must be met:

- ◆ The text of the SQL statement must be identical to the SQL statement that has already been parsed. This includes whitespaces.
- ◆ References to schema objects in the SQL statements must resolve to the same object.
- ◆ Bind variables must match the same name and data type.
- ◆ The SQL statements must be optimized using the same approach; in the case of the cost-based approach, the same optimization goal.

You may think that these conditions make it difficult to take advantage of the shared SQL areas. But by reusing application code, you can quite easily meet these conditions. When writing applications, you should strive to use the same SQL statements to access the same data and ensure that these SQL statements can meet these criteria.

Use stored procedures and functions whenever possible to guarantee that the same shared PL/SQL area is used. Another advantage is that stored procedures are stored in a parsed form—eliminating runtime parsing altogether.

Standardizing on naming conventions for bind variables and spacing conventions for SQL and PL/SQL statements can also increase the likelihood of reusing shared SQL statements.

The V\$LIBRARYCACHE table contains statistics on how well you are using the library cache. The important columns to view in this table are PINS and RELOADS:

- ◆ *PINS*: The number of times the item in the library cache was executed.
- ◆ *RELOADS*: The number of times the library cache missed and the library object was reloaded.

A few number of reloads relative to the number of executions indicates a high cache-hit rate. To get an idea of the total number of cache misses, use this statement:

```
SQL> SELECT SUM(reloads) "Cache Misses",
  2  SUM(pins) "Executions",
  3  100 * ( SUM(reloads) / SUM(pins) ) "Cache Miss Percent"
  4  FROM v$librarycache;
```

Cache Misses	Executions	Cache Miss Percent
9	2017	.44620724

The preceding example indicates that a sum of 2,017 SQL statements, PL/SQL blocks, and object definitions were accessed and only 9 were reloaded because they had aged out of the library cache. This means that only 0.44 percent of these statements resulted in reparsing—a very good cache-hit ratio.

To look at the cache hits based on the types of statements, use the following statement:

```
SQL> SELECT namespace,
  2  reloads "Cache Misses",
  3  pins "Executions"
  4  FROM v$librarycache;
```

NAMESPACE	Cache Misses	Executions
SQL AREA	4	1676
TABLE/PROCEDURE	5	309
BODY	0	0
TRIGGER	0	0
INDEX	0	21
CLUSTER	0	15
OBJECT	0	0
PIPE	0	0

8 rows selected.

The total number of reloads should be near zero. If you see more than 1 percent library cache misses, you should take action: reduce the cache misses by writing identical SQL statements or by increasing the size of the library cache.

You should be able to reduce the library cache misses by increasing the amount of memory available for the library cache. Do this by increasing the Oracle tunable parameter `SHARED_POOL_SIZE`. You may also need to increase the number of cursors available for a session by increasing the Oracle parameter `OPEN_CURSORS`.

Be careful not to increase the amount of memory required beyond that set aside by the operating system. Any paging or swapping caused by that offsets any advantage you get from the library cache.

If you have plenty of memory, you may be able to speed access to the shared SQL areas by setting the Oracle initialization parameter `CURSOR_SPACE_FOR_TIME` to TRUE. When this parameter is set to TRUE, it specifies that a shared SQL area cannot be deallocated until all the cursors associated with it are closed.

If `CURSOR_SPACE_FOR_TIME` is TRUE, it is not necessary for Oracle to check to see whether the SQL statement is in the library cache because it cannot be deallocated as long as the cursor is open. If memory is scarce on your system, do not set this parameter. If the value is TRUE and there is no space in the shared pool for a new SQL statement, an error is returned, halting the application.

Now that you have reviewed the advantages of using stored procedures, functions, and packages, the following sections present some specifics on how to use them.

Procedures and Functions

Procedures and functions are similar. In fact, they are so much alike that, throughout this book (except for this chapter), they have been referred to indiscriminately as *stored procedures*. Procedures and functions are subprograms made up of PL/SQL code that take a set of parameters given to them by the calling program and perform a set of actions. The difference between a procedure and a function is that a function can include a return value. Both functions and procedures can modify and return data passed to them as a parameter. Usually, procedures are used unless only one return value is needed. A procedure or function that has been stored in the library cache is referred to as a *stored procedure* or a *stored function*.

A stored procedure or stored function has the following properties:

Property	Comments
Has a name	This is the name by which the stored procedure or function is called and referenced.
Takes parameters	These are the values sent to the stored procedure or function from the application.
Returns values	A stored procedure or function can return one or more values based on the purpose of the procedure or function.
Stored in data dictionary	The stored procedure or function is stored in a parsed form in the data dictionary.

Procedures

A *procedure* is a set of PL/SQL statements that form a subprogram. The subprogram is designed and created to perform a specific operation on data in your database. A procedure takes zero or more input parameters and returns zero or more output parameters. The syntax of a procedure is as follows:

```
PROCEDURE procedure_name [( parameter_declaration )] IS
    [ local declarations ]
BEGIN
    PL/SQL Statements
[ EXCEPTION
    Optional Exception Handler(s) ]
END [ procedure_name ];
```

In this syntax, the *parameter_declaration* has the following format:

```
parameter_name [ IN | OUT | IN OUT ] datatype
```

The parameter qualifiers have the following meanings:

<i>Qualifier</i>	<i>Description</i>
IN	This parameter is used as an input value only.
OUT	This parameter is used as an output value only.
IN OUT	This parameter is used as both an input and an output variable.

The procedure is made up of two parts: the declaration and the body of the procedure. The declaration begins with the keyword **PROCEDURE** and ends with the last parameter declaration. The body begins with the keyword **IS** and ends with the keyword **END**.

The declaration part is used to define which variables are passed to the procedure and which values are returned from the procedure back to the calling program. The body of the procedure is where the real work is done. The body is made up of the PL/SQL statements that perform the desired task.

Functions

A *function* is a set of PL/SQL statements that form a subprogram. The subprogram is designed and created to perform a specific operation on data in your database. A function takes zero or more input parameters and returns one output value. If more than one output value is required, a procedure should be used. The syntax of a function is as follows:

```
FUNCTION function_name [( parameter_declaration )] RETURN datatype IS
    [ local declarations ]
BEGIN
    PL/SQL Statements
[ EXCEPTION
    Optional Exception Handler(s) ]
END [ function_name ];
```

The *parameter_declaration* has the same format as it does with a procedure:

```
parameter_name [ IN | OUT | IN OUT ] datatype
```

The parameter qualifiers have the following meanings:

<i>Qualifier</i>	<i>Description</i>
IN	This parameter is used as an input value only.
OUT	This parameter is used as an output value only.
IN OUT	This parameter is used as both an input and an output variable.

As with the procedure, the function is made up of two parts: the declaration and the body. The declaration begins with the keyword `FUNCTION` and ends with `RETURN` statement. The body begins with the keyword `IS` and ends with the keyword `END`.

The declaration part is used to define which variables are passed to the function and which values are returned from the function back to the calling program. The body of the function is where the real work is done. The body is made up of the PL/SQL statements that perform the desired task.

The difference between a procedure and a function is the return value. A function has the return declaration as well as a `RETURN` function within the body of the function that returns a value. This `RETURN` function is used to pass a return value to the calling program. If you do not intend to return a value to the calling program, or you want to return more than one value, use a procedure.

NOTE: For the remainder of this chapter, the term *procedure* is used to refer to both procedures and functions.

How Procedures and Functions Operate

Procedures and functions use the same basic syntax in the program body with the exception of the `RETURN` keyword that can only be used by functions. The body itself is made up of PL/SQL blocks that perform the desired function and return the desired data to the calling program. The goal of the body of the procedure is both to minimize the amount of data to be transmitted across the network (to and from the calling program) and to perform these PL/SQL statements in the most efficient manner possible.

The PL/SQL Language

PL/SQL is a block-structured language offered by Oracle to facilitate the use of the Oracle RDBMS. PL/SQL has the following properties and features that can be used to aid in application development:

Feature	Description
Block structure	The PL/SQL language is a block-structured language that allows blocks to contain nested subblocks.
Block declarations	Each block can have its own declarations, which means that you can logically separate functions.
Variable declaration	Variables can be declared and used within a PL/SQL block.
Constant declaration	Constants can be declared and referenced within a PL/SQL block.
Conditional statements	PL/SQL allows for conditional processing with <code>IF...THEN...ELSE</code> , <code>WHILE...LOOP</code> , <code>FOR...LOOP</code> , <code>EXIT...WHEN</code> , and <code>GOTO</code> functions.

These features make PL/SQL a powerful SQL processing language. The use of PL/SQL has several major advantages over the use of SQL statements (in addition to stored procedures and functions). Among these are ease of use, portability, and higher performance.

The primary performance difference between PL/SQL and SQL is the fact that PL/SQL statements are transmitted to Oracle as a block of statements rather than as individual statements. In a network application, the additional overhead needed to transmit individual statements can be quite high. It takes very little more CPU and network resources to send a larger packet than it does to send a smaller one.

The RETURN Statement

In the declaration portion of a function, a `RETURN parameter` is used to declare the type of the return value. Later, in the body of the function, the `RETURN statement` is used to exit the function and return the specified value to the calling program. With a procedure, the `RETURN statement` can also be used, but not to return a value. In a procedure, the `RETURN statement` can be used only to exit the procedure. No values can be associated with the `RETURN statement` in a procedure.

The EXCEPTION Statement

In both procedures and functions, you can add optional exception handlers. These exception handlers allow you to return additional information based on certain conditions (such as no data found or some user-specified condition). By using exception handlers and allowing the stored procedure to notify you of some special conditions, you can minimize the amount of return-value checking that must be done in the application code. Because the work to determine that no data has been selected has already been done by the RDBMS engine, you can save on resources if you take advantage of this information.

The RDBMS_OUTPUT Package

To visually represent data selected within a stored procedure or function, you can use the RDBMS_OUTPUT package supplied by Oracle. To see data returned by RDBMS_OUTPUT in SQL*Plus or Server Manager, you must set the SERVEROUTPUT option. When using the RDBMS_OUTPUT package, you can select several options for inputting or outputting data. The following procedures are available in the RDBMS_OUTPUT package:

Procedure	Description
RDBMS_OUTPUT.ENABLE	Enables output processing.
RDBMS_OUTPUT.DISABLE	Disables output processing.
RDBMS_OUTPUT.PUT_LINE	Places a newline-terminated string in the buffer.
RDBMS_OUTPUT.PUT	Places a string in the buffer (no newline).
RDBMS_OUTPUT.GET_LINE	Gets one line from the buffer.
RDBMS_OUTPUT.GET_LINES	Gets an array of lines from the buffer.

In this manner, you can use a stored procedure for ad-hoc functions that require data to be displayed in SQL*Plus. The typical stored procedure is used to return data that has been bound to variables in a program.

How To Create Stored Procedures and Stored Functions

There are advantages to using procedures and functions; however, the greatest advantage of using functions and procedures happens when the procedures and functions are stored in the database. Such procedures and functions are referred to as *stored procedures* and *stored functions*. A stored procedure or stored function has the advantage of being stored in the library cache in an already-parsed form, thus reducing parsing time.

To create a stored procedure or function, use the keywords CREATE PROCEDURE or CREATE FUNCTION with the same syntax as the PROCEDURE and FUNCTION commands shown earlier in this chapter. When creating a procedure or function, however, the IS keyword is replaced with the AS keyword. The following code is an example of how to create a stored procedure to retrieve some information from the DOGS table.

NOTE: The typical stored procedure or function is called by an application program. In the following example, however, to make it easier to show a stored procedure, I chose to use SQL*Plus.

```

SQL> CREATE OR REPLACE PROCEDURE
  2      old_dogs
  3  AS
  4      CURSOR dog_cursor IS
  5          SELECT
  6              dogname, age, owner
  7          FROM dogs
  8          WHERE age > 8;
  9  BEGIN
10      RDBMS_OUTPUT.PUT_LINE('Dogs older than 8 years old');
11      RDBMS_OUTPUT.PUT_LINE('Name    Age    Owner');
12      FOR dog IN dog_cursor LOOP
13          RDBMS_OUTPUT.PUT_LINE(dog.dogname||' '||dog.age||' '||dog.owner);
14      END LOOP;
15  END old_dogs;
16 /

```

Procedure created.

To view the output of this stored procedure from SQL*Plus, you must enable the SERVEROUTPUT option as follows:

```
SQL> set serveroutput on
```

The resulting output of this procedure is shown here:

```

SQL> execute old_dogs;
Dogs older than 8 years old
Name    Age    Owner
Shasta  9     Jones
Jessy   10    Wilson
Ruff    9     King

```

PL/SQL procedure successfully completed.

As you can see, to enable the stored procedure to return multiple rows selected from the DOGS table, it was necessary to declare a cursor. By looping through this cursor, you can output all the lines that were selected.

How To Replace Procedures and Functions

If the procedure or function is already stored in the library cache, you must *replace*, rather than *create*, the procedure or function. You do this by using the command CREATE OR REPLACE PROCEDURE or CREATE OR REPLACE FUNCTION. With this command, an already-present procedure or function is replaced; if it is not already present, it is created.

Packages

Packages are sets of related procedures or functions that are compiled and stored together in the data dictionary. Packages allow you to group together PL/SQL types, objects, and subprograms into a logical unit. When you link these logically related entities together, it can be

easier to program and modify modules based on their function and relation. Performance is enhanced because the entire package is loaded into memory when it is first called, thus increasing the chance for a cache hit on a related function or object that is likely to be called soon.

Packages are actually created in a statement with two different parts. The first is the declaration part, where the package is defined. Then there is the package body definition, where the body of the package is defined. The syntax of the statement used to create the package definition is as follows:

```
CREATE PACKAGE package_name AS package_specification
    public type and object declaration
    subprogram definition
END [ package_name ];
```

The definition part of the package creation declares the parts of the package available to the user. The rest of the package definition is used by the user, but is not visible to the user. This second part has the following syntax:

```
CREATE PACKAGE BODY package_name AS package_body
    private type and object declaration
    subprogram bodies
[ BEGIN
    initialization statements ]
END [ package_name ];
```

The user application has knowledge of the package specification. The arrangement of the package-creation process has several advantages:

<i>Advantage</i>	<i>Comment</i>
Portability	The body of the package can change without requiring any changes to the application—as long as the package specification does not change.
Security	The package can access tables you may not want the user to see. Because the package body is hidden from the user, some security can be maintained.
Modularity	With packages, modules can have specific functions that can be logically grouped and specified.
Ease of design	With packages, the specification part of the package can be completed first, thus allowing different teams to work on the package body and the application. Once the specification is completed, both groups can write to that specified interface.
Better performance	Because the entire package is loaded into memory when the first component is accessed, additional calls to the package do not invoke disk I/O.

Summary

As you have seen, the use of functions, procedures, and packages can improve performance in several ways. These performance enhancements include reduction in the amount of data that must be transmitted across the network and an increase in hits in the data dictionary cache.

Because a procedure, function, or package is stored within the data dictionary, it is available for immediate use by your applications. Because stored procedures and functions are stored in the library cache in an already-parsed form, performance is improved. Any set of SQL statements that your application frequently uses can benefit from being made into a stored procedure or function.

There are very few SQL statements that cannot benefit from the use of procedures, functions, and packages. By storing these subprograms in the database, you reduce network traffic and increase performance in accessing these programs. Whenever possible, use stored procedures and packages; there is no disadvantage associated with their use.

Chapter 29

Providing for Data Integrity and Triggers

When designing an application or a database, it is important that you design not only for performance and functionality but for data integrity as well. A database that does not guarantee data integrity is not worth having. The database and application are only as good as the data stored within them.

An accounting application that does not guarantee that the books balance does not have much value to the accountant who uses it. It is important to guarantee the integrity of your database. By placing integrity constraints on your database to enforce business rules, you guarantee that these rules are not violated.

It is very difficult to foresee every possible event in an application; it is even more difficult to foresee the outcome of ad-hoc changes to the database. By using integrity constraints, your data will be protected. This chapter looks at some ways you can protect your data—and how to do this in an optimal manner.

Integrity Constraints

It is usually necessary to enforce business rules in your application and database to protect your data. By using the Oracle integrity constraints, you can protect your data efficiently. Enforcing business rules from within Oracle can be more efficient than enforcing the same rules within your application for several reasons:

- ◆ **SQL statements can be reduced.** By allowing Oracle to enforce these business rules with integrity constraints, the application does not have to issue additional SQL statements. This reduces application overhead and network traffic.
- ◆ **Internal operations are faster.** Because integrity constraints are internal operations, they are naturally faster and more efficient.
- ◆ **Application development is simplified.** When you enforce the business rules within the database, multiple applications or parts of the same application need not duplicate the logic of the business rules.

This is not to say that the entire application has to rely on internal Oracle operations for data validations and bounds checking. In some cases, the application itself can handle these operations more efficiently; in general, however, Oracle integrity constraints are more efficient.

The following sections explain how Oracle integrity constraints can be used to enforce business rules and ensure data integrity.

Referential Integrity

Possibly the most common use of constraints is to enforce referential integrity. *Referential integrity* is used to guarantee that a column value that references another table exists in the other table. Referencing a value in another table that does not exist can cause major data integrity problems. Consider the following example.

The DOGS table introduced in earlier chapter contains a column that describes the breed of the dog (see Table 29.1). This value is actually a numeric value that references a row in the BREEDS table (see Table 29.2). If the DOGS table references a value for the dog's breed that does not exist in the BREEDS table, an unknown result occurs.

Table 29.1 The DOGS Table

<i>DOGNAME</i>	<i>AGE</i>	<i>BREED</i>	<i>OWNER</i>
Dash	6	1	Jones
Chip	4	1	Jones
Rigby	3	3	Smith
Duncan	5	2	Miller
Rufus	5	3	Smith
Splash	3	4	Blake
Piper	6	1	Turner
Pierce	6	1	King
Shasta	9	5	Adams
Teller	1	1	King
Spots	3	2	Ward
B.J.	3	4	Allan
Ginger	4	5	Turner
Jessie	10	7	Wilson
Ruff	9	8	King
Velvet	8	9	Wilson
Ty	4	10	Jones
Cotton	6	8	Atkins
Angel	6	8	McArthur
Provo	5	6	Smith
Jenny	7	6	Durell
Daphne	6	7	Bench
Bug	3	5	Durell
Sammie	8	3	Turner
Bubba	3	4	Pike

Table 29.2 The BREEDS Table

BREED	BREED NAME	DESCRIPTION
1	Border Collie	Very intelligent and happy
2	Sheltie	Energetic and willing to please
3	Jack Russell Terrier	High energy and active
4	Golden Retriever	Loves to play
5	All American (Mix)	Anything at all
6	Great Pyrenees	Big and friendly
7	Irish Setter	Hunting instinct
8	Toy Poodle	Lap dog
9	Beagle	Good nose for tracking
10	Greyhound	Fast and sleek

By applying referential integrity constraints to the DOGS and BREEDS tables, you can guarantee that the DOGS table does not reference any value for a dog's breed that does not exist. This constraint is applied by first adding BREED as a primary key in the BREEDS table:

```
SQL> ALTER TABLE breeds
  2      ADD PRIMARY KEY (breed);
```

Table altered.

Then you add BREED as a foreign key in the DOGS table:

```
SQL> ALTER TABLE dogs
  2      ADD FOREIGN KEY (breed) REFERENCES breeds(breed);
```

Table altered.

After you add these constraints, any attempt to add an additional entry to the DOGS table that references a nonexistent value in the BREEDS table results in the following error:

```
SQL> INSERT INTO dogs
  2  ( dogname, age, breed, owner )
  3  VALUES
  4  ( 'Lacy', 5, 12, 'Letterman' );
INSERT INTO dogs

ERROR at line 1:
ORA-02291: integrity constraint (ED.SYS_C00392) violated - parent key not found
```

By correcting the error, the `INSERT` operation will succeed:

```
SQL> INSERT INTO dogs
  2  ( dogname, age, breed, owner )
  3  VALUES
  4  ( 'Lacy', 5, 2, 'Letterman' );
1 row created.
```

By using referential integrity constraints, you reduce the chance of errors and ensure the integrity of your data. Some types of data checking (such as verifying that a field is numeric or verifying that all fields are entered) may be better done in the application code; however, in most other cases, checking is handled more efficiently by Oracle. A good rule of thumb is that if you have to issue an SQL statement to verify referential integrity, you can better protect your data with a referential integrity constraint.

Integrity Constraints

By using the features of the database, you can reduce processing at the application or the database level. Some integrity constraints can verify that your data conforms to requirements you set up in your tables. Following is a list of a few of these constraints:

- ◆ **NOT NULL.** The NOT NULL constraint specifies that a column must have a value associated with it. This is useful when the absence of a value can cause data integrity problems. This constraint is sometimes used with the UNIQUE constraint to ensure not only that the value exists and is valid, but that the value is unique.
- ◆ **UNIQUE.** The UNIQUE key constraint is typically used in the definition of the table's primary key. The UNIQUE key constraint specifies that the values inserted into the column or columns must be unique. Unique keys differ from primary keys in that they are not used to uniquely identify a row in the database, but rather to avoid duplications within the column. NULL values can be inserted unless the NOT NULL constraint is also present for that column.
- ◆ **PRIMARY.** The primary key in a table is used to uniquely identify a row in a table and to avoid duplication in the column or columns designated as the primary key. Although a primary key can be a composite of more than one row, this approach is discouraged. Because the PRIMARY key is used to identify the row in the table, it should be unique and somewhat stable. The values in the column used as the primary key should not change, contain NULLs, or be long. Short numeric values are ideal for primary keys; sequences are typically used to provide the key values.
- ◆ **FOREIGN.** The foreign key in a table is used to reference a primary key in another table and is used to maintain the relationship between these two tables. When you reference a primary key in another table as a foreign key, you maintain the relationship as well as the integrity of the data.
- ◆ **CHECK.** The CHECK constraint can be used to enforce business rules that cannot be enforced by any other type of constraint. For example, this type of business rule may be checking that a value does not exceed a maximum value or fall below a minimum value. Other examples of nonstandard business rules are checking that a column value is not larger than another column value or that only certain discrete values are used. You can use the CHECK constraint to handle these kinds of miscellaneous functions.

Using Constraints

Constraints can be implemented at one of the following times:

- ◆ **Table creation.** When the table is created and the columns are defined, the integrity constraints can be specified.
- ◆ **Table modification.** Constraints can be added to already existing tables by using the ALTER TABLE command.

Implementing Constraints at Table Creation

When creating tables, you can add constraints to the column definitions as shown in the following CREATE TABLE command:

```
CREATE TABLE breeds
( breed NUMBER(4) PRIMARY KEY,
  breed_name VARCHAR2(25) NOT NULL,
  description VARCHAR2(40) NOT NULL)
TABLESPACE dog_space;

CREATE TABLE dogs
( dogname VARCHAR2(40) NOT NULL,
  age NUMBER(2) NOT NULL
    CONSTRAINT age_check
      CHECK ( age < 20 ),
  breed NUMBER(4)
    CONSTRAINT breed_cons
      REFERENCES breeds ON DELETE CASCADE,
  owner VARCHAR2(40) NOT NULL )
TABLESPACE dog_space;
```

When specifying referential integrity constraints, you can specify two additional options:

<i>Option</i>	<i>Definition</i>
ON DELETE CASCADE	Specifies that if rows in the parent table are deleted, the rows referenced in the child table are also deleted.
No Action	By not specifying the ON DELETE CASCADE qualifier, no changes are allowed to parent table columns that have values in child tables that reference them.

Implementing Constraints on Existing Tables

For existing tables, you can add constraints with the ALTER TABLE command, as shown here (all three commands are required to alter the verification methods used by the DOGS and BREEDS tables introduced earlier in this chapter):

```
ALTER TABLE breeds
  ADD PRIMARY KEY (breed);
```

```

ALTER TABLE dogs
  ADD FOREIGN KEY (breed) REFERENCES breeds(breed);

ALTER TABLE dogs
  ADD CHECK (age < 20);

```

Viewing Constraints

When you add constraints to tables, these definitions are added to the data dictionary. The constraint information is kept internal tables in the data dictionary. These tables can be queried in several views. The following chart lists the views defined for constraint information.

<i>View</i>	<i>Which Constraints?</i>
ALL_CONSTRAINTS	Constraints on objects accessible by the user.
USER_CONSTRAINTS	Constraints on objects owned by the user.
DBA_CONSTRAINTS	Constraints on all objects in the system.
ALL_CONS_COLUMNS	Column information about constraints on objects accessible by the user.
USER_CONS_COLUMNS	Column information about constraints on objects owned by the user.
DBA_CONS_COLUMNS	Column information about constraints on all objects in the system.

You can obtain information about which tables have constraints applied to them by querying the data dictionary in this manner:

```

SQL> SELECT
  2    constraint_name,
  3    constraint_type,
  4    table_name,
  5    r_constraint_name
  6   FROM
  7    user_constraints;

CONSTRAINT_NAME          C TABLE_NAME          R_CONSTRAINT_NAME
-----  -----  -----
SYS_C00388                C BREEDS
SYS_C00389                C BREEDS
SYS_C00390                C BREEDS
SYS_C00391                P BREEDS
SYS_C00378                C DOGS
SYS_C00379                C DOGS
SYS_C00380                C DOGS
SYS_C00381                C DOGS
SYS_C00392                R DOGS
SYS_C00404                C DOGS
                                         SYS_C00391

```

10 rows selected.

Alternatively, you can obtain specific constraint information in this way:

```
SQL> SELECT
  2    constraint_name,
  3    search_condition
  4  FROM
  5    user_constraints
  6 WHERE
  7    table_name = 'DOGS';

CONSTRAINT_NAME
-----
SEARCH_CONDITION
-----
SYS_C00378
DOGNAME IS NOT NULL

SYS_C00379
AGE IS NOT NULL

SYS_C00380
BREED IS NOT NULL

SYS_C00381
OWNER IS NOT NULL

SYS_C00392

SYS_C00404
age < 20
```

6 rows selected.

To get column information, you can use the following statement:

```
SQL> SELECT
  2    constraint_name,
  3    table_name,
  4    column_name
  5  FROM
  6    user_cons_columns;

CONSTRAINT_NAME          TABLE_NAME        COLUMN_NAME
-----
SYS_C00378              DOGS             DOGNAME
SYS_C00379              DOGS             AGE
SYS_C00380              DOGS             BREED
SYS_C00381              DOGS             OWNER
SYS_C00388              BREED            BREED
SYS_C00389              BREED            BREED_NAME
SYS_C00390              BREED            DESCRIPTION
SYS_C00391              BREED            BREED
SYS_C00392              DOGS             BREED
SYS_C00404              DOGS             AGE
```

10 rows selected.

Review of Integrity Constraints

Constraints can be a very powerful tool for protecting the integrity of your data. By using Oracle integrity constraints, you can simplify the application development process and gain additional efficiency by using internal Oracle functions. If constraints and business rules are needed in your application and for your database, it is much more efficient to use the Oracle integrity constraints rather than coding the validation procedures yourself.

Triggers

Another way to ensure and verify data integrity is with the use of triggers. A *trigger* is a procedure stored in the database; it is *fired* whenever a table is modified. Triggers can be used to enforce integrity constraints and database security.

A trigger is created and defined to fire either before or after a modification to the table has occurred. To create a trigger, use the Oracle `CREATE TRIGGER` command. By creating triggers on database objects, you can further ensure data integrity.

Although triggers are typically used to enforce referential integrity, there are other uses for triggers:

- ◆ **Auditing.** Triggers can be used to create audit information for certain types of transactions or table accesses.
- ◆ **Data validation.** By using triggers, you can perform additional data validation to prevent invalid data from being inserted into the database.
- ◆ **Security.** Triggers can be used to enforce security rules.
- ◆ **Data generation.** Data made up of other data can be generated with triggers.
- ◆ **Enforce business rules.** Complex business rules can be enforced by using triggers.

Using Triggers

The preceding list gave several uses of triggers. The following list shows a few ways triggers can be used in applications:

- ◆ **Event logging.** Triggers can be used to automatically log accesses to certain tables in the database. By using triggers to log accesses on a table basis, you can individually monitor each table and retrieve the results of such monitoring. Remember that although Oracle offers auditing features with the `AUDIT` command, if your needs are very specific and limited to certain tables, triggers can be more efficient.

- ◆ **Data validation.** Triggers can be useful for data validation when the use of referential integrity constraints are not sufficient. The use of triggers can ensure that invalid transactions are not executed on the database.
- ◆ **Data generation.** Triggers can also be used to generate derived data. By firing a trigger on certain input data, other data derived from that input data can be automatically generated. Triggers can facilitate the creation of the data and can provide performance enhancements over generating this data in the application. Performance is enhanced because network traffic can be reduced as can the overhead involved in issuing additional SQL statements necessary to perform the same action.

Using Alerts

When using triggers, you can take advantage of the RDBMS_ALERT package supplied by Oracle. With this package, you can enable the use of asynchronous notification of database events. With the DBMS_ALERT package, applications can be signaled if an event occurs in the database—a useful function for applications that need an event to occur before proceeding or for applications that signal error conditions that have occurred.

Waiting for events to occur asynchronously is much more efficient than polling the database to see whether the event has occurred. When you poll, you generate network traffic and execute SQL statements unnecessarily. If your application must wait for an event, using the RDBMS_ALERT package is the way to do it.

Creating Triggers

Triggers are created in a manner almost identical to stored procedures. The syntax for a trigger begins with these keywords:

```
CREATE OR REPLACE TRIGGER trigger_name
```

These keywords are followed by the WHEN clause, which determines when the trigger is fired. The WHEN clause can consist of these keywords:

BEFORE or AFTER

Following these keywords are the qualifying statement(s):

```
DELETE or INSERT or UPDATE
```

The qualifying statements are followed by this clause:

```
ON table_name
```

In the case of an UPDATE, you can specify column names.

The addition of this statement specifies that the trigger will fire for each row that is accessed, rather than on a statement basis:

FOR EACH ROW

Here is an example of a trigger that uses the familiar DOGS table:

```
SQL> CREATE OR REPLACE TRIGGER old_one
  2    AFTER INSERT OR UPDATE OR DELETE ON dogs
  3    FOR EACH ROW
  4    BEGIN
  5      IF (:new.age > 8) THEN
  6        RDBMS_OUTPUT.PUT_LINE('Note: Dog is older than 8 years.');
  7      END IF;
  8    END;
  9  /
```

Trigger created.

Now if you insert a dog into the DOGS table that is older than 8 years old, you get the following output:

```
SQL> INSERT INTO dogs
  2  ( dogname, age, breed, owner )
  3  VALUES
  4  ( 'Molly', 14, 5, 'Hanks' );
Note: Dog is older than 8 years.

1 row created.
```

Viewing Triggers

When triggers are added to tables, the definitions are added to the data dictionary. The constraint information is kept in internal tables in the data dictionary. These tables can be queried from several views. The views defined for constraint information are as follows:

<i>View</i>	<i>Which Triggers?</i>
ALL_TRIGGERS	Triggers defined on objects accessible by the user.
USER_TRIGGERS	Triggers defined on objects owned by the user.
DBA_TRIGGERS	Triggers defined on all objects in the system.

In these tables, some of the important information concerning triggers is contained in the following columns:

- ◆ *TRIGGER_NAME*: The identifier of the trigger.
- ◆ *TRIGGERING_EVENT*: What event “fires” the trigger.

- ◆ *TABLE_NAME*: The name of the table that the trigger is set on.
- ◆ *WHEN_CLAUSE*: When the trigger fires.
- ◆ *TRIGGER_BODY*: The code that is run when the trigger fires.

This information can be useful when analyzing or debugging triggers.

Review of Triggers

Triggers can be very useful not only as a supplement to Oracle integrity constraints but as a way to provide for additional auditing, data verification, and data generation. By using triggers for these purposes, you can simplify and enhance your application code. By reducing the number of SQL statements that must be called by the application and allowing internal functions to provide the features, you improve efficiency.

Triggers are frequently used to enforce business rules that cannot be controlled by Oracle integrity constraints and as supplements to the Oracle auditing facilities.

Triggers can also be used as an efficient way of logging access to specific tables.

Because triggers use significant CPU resources, you should use them sparingly—but take advantage of them if you need them.

Audit Trails

Oracle has built-in auditing facilities you can use to create an *audit trail* of events that have occurred in the RDBMS. The audit trail is somewhat configurable and can contain as much or as little auditing information as you want. By increasing the number of events to be audited, you increase the overhead incurred by the RDBMS. It is wise to audit as little as you can get away with to reduce the impact on the system.

Audit trail information can contain the following:

- ◆ The user accessing the object
- ◆ The name of the object accessed
- ◆ The operation performed or attempted
- ◆ The timestamp of that operation
- ◆ The session and terminal identifiers

Use the `AUDIT` and `NOAUDIT` commands to turn auditing on and off on a per-object basis. The procedure for doing this, and the objects that can be modified, are explained in the Oracle documentation and are not covered here.

Auditing is an important part of the security of your system, but you should limit it to only the most critical objects. Auditing involves significant use of system resources; by reducing the amount of auditing you do, you can lessen the impact on the system.

If you are in a somewhat secure environment, you may not find auditing necessary. But if you are in a large organization with lots of activity on the RDBMS, you should seriously consider auditing. What to audit is a decision to be made on a case-by-case basis and cannot be taken lightly. As much as possible, limit your auditing to reduce overhead, but audit as much as you need to.

If auditing is being done, the auditing information is kept in the AUD\$ table. Check this table periodically to make sure that it does not fill up the SYSTEM tablespace.

Serial Reads

If your application requires serializable reads, you must change the parameters `SERIALIZABLE` and `ROW_LOCKING` in the Oracle initialization file. By changing these parameters, you alter the performance of the system dramatically. Oracle recommends that users implement applications using the default row locking.

Oracle has put considerable effort into improving the performance of serializable transactions to compete with other RDBMS vendors in the TPC-C benchmark and has done very well. This does not mean that application developers are forced to use serializable transactions. Oracle row-level locking is suitable for almost every application.

By setting the Oracle initialization parameter `SERIALIZABLE` to TRUE, queries acquire table-level read locks, preventing updates to the objects being read until the transaction containing the query has been committed. This mode provides for repeatable reads and ensures that two queries for the same data in the same transaction see the same data.

This parameter provides for ANSI-degree-three consistency but at a considerable cost in concurrency.

Summary

This chapter presented several ways you can optimally provide for data integrity. When designing applications, you should take advantage of Oracle features such as integrity constraints and triggers where applicable. By using internal features such as integrity constraints, you can reduce unnecessary overhead in the application, the network, and the server.

It is important that you do not sacrifice integrity for performance. A database that does not guarantee data integrity is not worth having. The database and the application are only as good as the data stored within them.

Another topic covered in this chapter was the use of auditing. Although auditing may or may not be an important part of your operation, by taking advantage of only the auditing features you need and eliminating those you don't need, you can optimize performance and still track required information.

Finally, the chapter briefly discussed the use of serializable reads, which some applications need for data integrity. I believe that a well-designed application that understands Oracle row locking can perform well and not sacrifice data integrity. The use of serializable transactions does not provide *more* data integrity, it just provides it in a different way than row locking. If at all possible, avoid the use of serializable reads in your database and application.

Chapter 30

Using Hints

The Oracle optimizer is very efficient and works quite well to produce the best execution plan for your SQL statements based on the information it has to work with. The optimizer does not, however, have the amount of information about your database and your data as you do. This is why Oracle allows you to use hints to tell the optimizer what kind of operations will be more efficient based on knowledge you have about your database and your data.

By using hints, you can tell the optimizer such things as these:

- ◆ The best optimization approach for a particular SQL statement
- ◆ The goal of the cost-based approach for an SQL statement
- ◆ The access path for a statement
- ◆ When table scans are more efficient than indexes
- ◆ The join order for a statement
- ◆ A join operation in a join statement
- ◆ The degree of parallelism in a parallel query statement

By using hints, you can use specific information that you know about your data and database to further enhance the performance on certain SQL statements. With hints, you can enhance specific operations that might otherwise be inefficient. Here are some examples of conditions in which hints may significantly improve performance:

- ◆ **Indexed columns with a large number of duplicate values.** Telling the optimizer to bypass the index when the value is one you know has a large number of duplicates is more efficient than letting the optimizer use the index.
- ◆ **Table access that performs a large table scan.** By specifying a larger number of parallel query servers, you can improve performance.
- ◆ **Table access that performs a small table scan.** If you know that the amount of data to be scanned is small, you may want to disable the parallel query option for this particular operation.

These are just a few of the exceptional conditions in which the default optimization may not be efficient. The information you know about your data and application can be used to make more efficient optimization choices.

This chapter looks at the different optimization hints available and when you can use them to improve the optimization of your SQL statements. The chapter first explains how hints are implemented within your SQL statements and then describes the hints themselves, categorized by function.

Implementing Hints

Hints are implemented by enclosing them within a comment to the SQL statement. An SQL statement can have only one comment containing hints; these hints can follow only the SELECT, UPDATE, or DELETE keyword. Each hint applies only to the statement block in which the hint appears.

Hint Syntax

If you have a compound query in which several `SELECT`s are combined with the `UNION` operator, each query must contain its own hint. The hint applied to one of the `SELECT` statements does not carry over to any other `SELECT`. Oracle comments can have two forms:

<code>/* comment */</code>	The <code>/*</code> indicates the beginning of the comment; the <code>*/</code> indicates the end of a comment. The comment can be multiple lines long.
<code>-- comment</code>	The comment consists of the remainder of the line following the <code>--</code> characters.

Adding the `+` character following the beginning of the comment indicates to the parser that the following text is a hint to the optimizer so that it is parsed as such (refer to the example in the following section).

Hint Errors

If a hint is incorrectly specified, Oracle ignores it and does not return an error. The following conditions cause hints to be ignored:

- ◆ Comment does not follow a `SELECT`, `UPDATE`, or `DELETE` statement. If the comment containing the hint does not follow one of these statements, the hint is ignored.
- ◆ Syntax errors. A hint containing syntax errors is ignored. Other hints within the same comment are evaluated individually.
- ◆ Conflicting hints. Conflicting hints (for example, `FULL` and `INDEX`) are ignored, but other nonconflicting hints within the same comment are evaluated individually.

Some Examples of Hints

Here is an example of how a hint in an SQL statement looks:

```
SELECT /*+ CHOOSE */ account_no, name, balance FROM bank_account WHERE
account_no = '12631';
```

This hint specifies that the optimizer should use the `CHOOSE` option, overriding any other optimization mode that might be set.

Here is another example of a hint (notice that the statement is the same; to use the `--` form of the comment, the comment and hint must be at the end of the line):

```
SELECT --+ CHOOSE
       account_no, name, balance
  FROM
    bank_account
 WHERE
   account_no = '12631';
```

The parser and optimizer consider both of these methods for indicating hints to be equivalent.

Using Multiple Hints

Except in the case of conflicting hints, it is possible to merge multiple hints together. In some cases, merging hints can be very useful and possibly essential.

In many parallel query hints, it is not uncommon for the **FULL** (full table scan) hint to be used in conjunction with the **PARALLEL** (parallel query) hint. By using these two hints together, you ensure that a full-table scan will be executed and that the parallel-table scan feature of the Parallel Query option will be used.

To use multiple hints together, simply add additional hints within the same comment, as in this example that combines the **FULL** and **PARALLEL** hints:

```
SELECT /* FULL( dogs ) PARALLEL ( dogs, 5 ) */
       dogname, age, owner
  FROM
    dogs
 WHERE
   age < 5;
```

Hints

The following sections describe the hints available to the Oracle optimizer. The hints are grouped into several categories based on function.

Optimization Approaches

The following hints allow you to indicate to Oracle to use a certain optimization approach, regardless of what optimization approach has been specified in the Oracle initialization file. The indicated approach takes precedence over any other specification of the optimization approach.

ALL_ROWS

The `ALL_ROWS` hint specifies that the cost-based optimizer should be used with the goal of best throughput. This method creates an execution plan with the least amount of total resource consumption and best throughput. The cost-based optimization approach is used, regardless of the presence or absence of statistics.

The syntax of this hint is as follows:

```
/*+ ALL_ROWS */
```

If you include hints for access paths or join operations with the `ALL_ROWS` hint, the optimizer gives precedence to the access paths and join hints.

If no statistics are available for the tables involved, and a hint forces the cost-based optimizer, the optimizer uses default statistics to determine the most optimal execution plan. These default statistics include allocated storage and number of data blocks used. These statistics are not very good, but they are better than no statistics at all. If you intend to use the cost-based optimizer, you should run the `ANALYZE` command on the tables, indexes, and clusters to be accessed to get better statistics.

CHOOSE

The `CHOOSE` hint causes the optimizer to choose the rule-based or cost-based optimization approach based on the presence of statistics available on the tables being accessed by the SQL statements. If statistics are available for the tables being accessed, the optimizer chooses the cost-based approach, with the goal of best throughput. If no statistics exist for the tables being accessed, the rule-based optimizer is used.

The syntax of this hint is as follows:

```
/*+ CHOOSE */
```

FIRST_ROWS

The `FIRST_ROWS` hint specifies that the cost-based optimizer should be used with the goal of best response time. This method creates an execution plan with the least amount of resource consumption to return the first row. The cost-based optimization approach is used, regardless of the presence or absence of statistics.

The syntax of this hint is as follows:

```
/*+ FIRST_ROWS */
```

When you use the `FIRST_ROWS` hint, the optimizer also makes the following choices:

- ◆ If an index scan is available, it may be chosen over a full-table scan.

- ◆ If an index scan is available, the optimizer may choose a nested-loops join over a sort-merge join whenever the associated table is the potential inner table of the nested loops.
- ◆ If an index scan is available through an `ORDER BY` clause, the optimizer may choose it to avoid a sort operation.

The `FIRST_ROWS` hint is ignored in `DELETE` and `UPDATE` statements that contain the following keywords:

- ◆ Set operators (`UNION`, `INTERSECT`, `MINUS`, `UNION ALL`)
- ◆ `GROUP BY` clauses
- ◆ `FOR UPDATE` clauses
- ◆ Group or aggregate functions such as `AVG`, `MIN`, `MAX`, `COUNT`, `SUM`, `STDDEV`, and `VARIANCE`
- ◆ `DISTINCT` operator

These statements cannot be optimized for best response time by retrieving the first row because the operation involved requires that all rows must be retrieved before returning the first row. If you do use the `FIRST_ROWS` hint in any of these statements, the cost-based optimizer is still used, regardless of the presence of statistics on the tables involved.

If you include hints for access paths or join operations with the `FIRST_ROWS` hint, the optimizer gives precedence to the access paths and join hints.

If no statistics are available for the tables involved and a hint forces the cost-based optimizer, the optimizer uses default statistics to determine the most optimal execution plan. These default statistics include allocated storage and number of data blocks used. These statistics are not very good, but they are better than no statistics at all. If you intend to use the cost-based optimizer, you should run the `ANALYZE` command on the tables, indexes, and clusters to be accessed to get better statistics.

RULE

The `RULE` hint specifies that the rule-based optimization approach should be used. This hint also causes the optimizer to ignore any other hints that may be specified in this statement block.

The syntax of this hint is as follows:

```
/*+ RULE */
```

Because the rule-based approach is based simply on the SQL statements themselves, you must have statistics about the database tables. The rule-based approach uses the following steps to determine the execution plan:

1. Determines possible execution plans.
2. Ranks the different plans according to a specific ranking (see Table 27.1 in Chapter 27, “Using the Oracle Optimizer”).
3. Chooses the approach with the lowest ranking.

In this way, the rule-based optimization approach is very efficient and works well. However, if statistics are available for your tables, clusters, or indexes, the cost-based approach can be very efficient.

Access Methods

The following hints provide the optimizer with the specific access method you think is more appropriate, based on information you know about your data and your application. These hints are valid only if the referenced index or cluster is available and the syntax of the SQL statement is correct. If the syntax is incorrect or the referenced index or cluster is not available, the hint is ignored.

When specifying an access path for an SQL statement, the table to be accessed must be specified exactly. If the table is accessed with an alias, you must specify the alias.

AND_EQUAL

The **AND_EQUAL** hint specifies an execution plan that uses an access path that merges the scans of several single-column indexes.

The syntax of this hint is as follows:

```
/*+ AND_EQUAL( table index index [ index .... index ] ) */
```

CLUSTER

The **CLUSTER** hint specifies that the optimizer should choose a cluster scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ CLUSTER( table ) */
```

FULL

The **FULL** hint specifies that the optimizer should choose a full-table scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ FULL( table ) */
```

This hint specifies the use of a full-table scan, even though there may be an index present on this table and the index key is specified in the **WHERE** clause.

TIP: If you have an indexed table and you know that a certain value in your **SELECT** statement has a large number of duplicates (which is inefficient for indexes), you may want to use the **FULL** hint to bypass the index.

HASH

The `HASH` hint specifies that the optimizer should choose a hash scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ HASH( table ) */
```

INDEX

The `INDEX` hint specifies that the optimizer should choose an index scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ INDEX( table index [ index .. index ] ) */
```

This hint may contain one or more indexes. The following choices may be made by the optimizer:

- ◆ If no index is specified in the hint, the optimizer determines the cost of each index that could be used to access the specified table. Once this determination is made, the index with the lowest cost is chosen for the index scan. The optimizer may choose to scan multiple indexes and merge the result if this path has the lowest cost. A full-table scan is not considered.
- ◆ If one index is specified in the hint, this index is used for the index scan. The optimizer does not consider a full-table scan or any other index scans.
- ◆ If multiple indexes are specified in the hint, the optimizer determines the cost of each index in the list that could be used to access the specified table. Once this determination is made, the index with the lowest cost is chosen for the index scan. The optimizer may choose to scan multiple indexes in the list and merge the result if this path has the lowest cost. A full-table scan is not considered, nor are any indexes not in the list.

The `INDEX` hint specifies the use of an index scan, even though the optimizer may not choose an index scan because of a low number of distinct values. If you know that the value in the `WHERE` clause is somewhat unique, you can improve performance by using this hint.

TIP: If you have an indexed table, and you know that a certain value in your `SELECT` statement has a small number of distinct values (which is inefficient for indexes), but the value you are selecting is somewhat distinct, you may benefit from this hint.

INDEX_ASC

The `INDEX_ASC` hint specifies that the optimizer should choose an index scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ INDEX_ASC( table index [ index .. index ] ) */
```

This hint can contain one or more indexes. The `INDEX_ASC` hint is similar to the `INDEX` hint except that if the statement specifies an index range scan, the entries are scanned in ascending order of their indexed values.

NOTE: This method is currently the default behavior for a range scan. However, because Oracle does not guarantee that this method will always be the default behavior, this hint is provided.

INDEX_DESC

The `INDEX_DESC` hint specifies that the optimizer should choose an index scan to access the specified table.

The syntax of this hint is as follows:

```
/*+ INDEX_DESC( table index [ index .. index ] ) */
```

The `INDEX_DESC` hint may contain one or more indexes. This hint is similar to the `INDEX` hint except that if the statement specifies an index range scan, the entries are scanned in descending order of their indexed values.

ROWID

As you know, you do not necessarily have to perform a table scan by row ID. If you want to do a table scan by row ID, use the `ROWID` hint to specify that the optimizer should choose a table scan by row ID to access the specified table.

The syntax of this hint is as follows:

```
/*+ ROWID( table ) */
```

This hint specifies the use of a table scan by `ROWID`, even though an index may be present for this table and the index key is specified in the `WHERE` clause.

USE_CONCAT

The `USE_CONCAT` hint forces combined `OR` conditions in the `WHERE` clause of a query to be transformed into a compound query using the `UNION ALL` set operator. Typically, this transformation is done only if the cost is cheaper with concatenation than without it.

The syntax of this hint is as follows:

```
/*+ USE_CONCAT */
```

Join Orders

The following hint is used to specify join orders that take advantage of information you know about your data to improve the performance of the joins.

ORDERED

The ORDERED hint caused the execution plan to join tables in the order they are specified in the FROM clause.

The syntax of this hint is as follows:

```
/*+ ORDERED */
```

Because you know your data better than the Oracle optimizer, you may be able to specify the order of the join to make better choices for the inner and outer tables than the optimizer can. The join operations mentioned in the following section can be used with the ORDERED hint.

Join Operations

The following hints are used to specify join operations and to take advantage of information you know about your data to improve the performance of the joins. When specifying a join operation for an SQL statement, the table to be joined must be specified exactly. If the table is accessed with an alias, you must specify the alias. The USE_MERGE and USE_NL hints must be used with the ORDERED hint.

USE_MERGE

The USE_MERGE hint causes the execution plan to join each specified table with another row source using a sort-merge join.

The syntax of this hint is as follows:

```
/*+ USE_MERGE ( table ) */
```

You can use this hint with the ORDERED hint as follows:

```
/*+ ORDERED USE_MERGE ( table ) */
```

In this syntax, *table* refers to a table to be joined to the row source that results from joining other tables in the join order using a sort-merge join.

USE_NL

The **USE_NL** hint causes the execution plan to join each specified table with another row source using a nested-loops join. The specified table is used as the inner table.

The syntax of this hint is:

```
/*+ USE_NL ( table ) */
```

In this syntax, *table* refers to the table to be used as the inner table of the nested-loops join.

Parallel Query Hints

The following hints are related to the Oracle Parallel Query option. As I have said many times in this book, I am a great fan of the Parallel Query option: you can achieve great results when you use it.

CACHE

The **CACHE** hint specifies that the blocks retrieved for the table in the hint are placed at the most-recently-used end of the LRU (least recently used) list in the buffer cache when a full-table scan is performed. When a full-table scan occurs, the entries are usually put on the end of the LRU to age more quickly. This is done because most of the data read in a full-table scan is usually discarded. The **CACHE** hint can be useful if most of the data in a full-table scan is used.

The syntax of this hint is as follows:

```
/*+ CACHE ( table ) */
```

Alternatively, you can use the **CACHE** hint with the **FULL** hint:

```
/*+ FULL( table ) CACHE ( table ) */
```

PARALLEL

The **PARALLEL** hint specifies the desired number of query servers to be used for this specific operation.

The syntax of this hint is as follows:

```
/*+ PARALLEL ( table [ , degree ] [ , split ] ) */
```

Alternatively, you can use the **PARALLEL** hint with the **FULL** hint:

```
/*+ FULL( table ) PARALLEL ( table [ , degree ] [ , split ] ) */
```

Typically, the *table* and *degree* parameters are specified.

This hint takes two comma-separated values after the table definition. The first value specifies the degree of parallelism; the second value specifies the split among instances in a parallel server environment. If *degree* is not specified, a value is chosen by the query coordinator.

NOCACHE

The **NOCACHE** hint specifies that the blocks retrieved for the table in the hint are placed at the least-recently-used end of the LRU (least recently used) list in the buffer cache when a full-table scan is performed. This is the default behavior in a table scan. The buffer entries are put on the end of the LRU to age more quickly. This is done because most of the data read in a full-table scan is usually discarded. The **NOCACHE** hint can be useful if very little of the data in a full table scan is used.

The syntax of this hint is as follows:

```
/*+ NOCACHE ( table ) */
```

Alternatively, you can use the **NOCACHE** hint with the **FULL** hint:

```
/*+ FULL( table ) NOCACHE ( table ) */
```

The **NOCACHE** hint can be useful if you think you will be reading large amounts of data that will unnecessarily be held in the buffer cache.

NOPARALLEL

The **NOPARALLEL** hint can be used to override the default parallel query operations, even though the default table parameters allow for parallel query processing.

The syntax of this hint is as follows:

```
/*+ NOPARALLEL ( table ) */
```

The **NOPARALLEL** hint can be useful for certain conditions in which you have to reduce the load on the system or reduce I/O contention. This hint essentially disables parallel query processing for this SQL statement.

PUSH_SUBQ

The **PUSH_SUBQ** hint causes nonmerged subqueries to be evaluated at the earliest possible place in the execution plan. Normally, subqueries that are not merged are executed as the last step in the execution plan. If the subquery is inexpensive and reduces the number of rows significantly, you can improve performance by executing it earlier.

The syntax of this hint is as follows:

```
/*+ PUSH_SUBQ */
```

Summary

Because you know more about your data and your application than the Oracle optimizer does, you can make significant improvements to the execution plan of the SQL statements. The Oracle optimizer is very efficient and works quite well to produce the best execution plan for your SQL statements based on the information it has to work with; however, anything you can do to give the optimizer additional information about the execution process will help performance.

By using hints, specific information you know about your data and database can be used to further enhance the performance of certain SQL statements. By using hints, you can enhance specific operations that might otherwise be inefficient. The best way you can significantly improve the performance of your system is by knowing it. Understand the data access patterns. Determine what the users are most likely to access and how they will access it.

By knowing your application and data, you can significantly improve the performance of your server and your application. Hints can help you transfer your knowledge of your data and application to the Oracle optimizer.

Chapter 31

Introducing SQL Development Tools

When building client applications, it is sometimes useful to employ Oracle or third-party tools to enhance the efficiency of these applications. These tools are available for many areas and can help with many different aspects of your application and database design.

Tools fall into several different categories:

- ◆ **Database design tools.** This type of tool helps in the design of the database itself. By feeding the tool an outline of the data to be stored in the database—as well as the data relationships—this type of tool designs the tablespaces, tables, clusters, and indexes.
- ◆ **Application builders.** This type of tool is used to actually build the client application. The tools typically have an emphasis on the Graphical User Interface (GUI).
- ◆ **Analysis tools.** These tools examine or trace your SQL statements, try to determine inefficiencies and bottlenecks, and recommend ways to correct those problems.

The following sections examine a few of these different types of tools and describe how they can help you optimize your database.

Database Design Tools

Designing a database can be a difficult task, even for the most seasoned Oracle users. When a database is made up of ten or more tables, each with relationships to other tables, the task can be even more difficult. Along with building the tablespaces, tables, clusters, and indexes is the complexity of deciding how to arrange the physical data files to most efficiently distribute the I/Os.

By taking advantage of some of the database design tools available on the market today, you can reduce some of this complexity. Database design tools allow you to input information about the data that must be stored in the database and the relationships between that data; they produce a recommended database layout you can take and modify. No matter how good the design tool, you must still make the final decisions and modifications.

These tools have come a long way in the last few years and have seen an incredible amount of improvement. Even so, you must be the final judge of the design of your database. Take these tools, use them, and modify the output to suit your needs. Database design tools are available from several vendors including Oracle. Oracle includes this type of design tool as part of its Designer/2000 package. Other tools include S-Designor from SDP Technologies, ERwin/ERX from Logic Works, and SQL Coder from Platinum Technologies.

Oracle Designer/2000

The Oracle Designer/2000 is a tool developed by Oracle to assist the corporate customer in designing and implementing an information management system. Designer/2000 is based on rapid development of information systems that are business driven. Designer/2000 is intended to help businesses develop the information system they need to re-engineer the way they do business.

Designer/2000 is repository based and is designed to allow users to share work; a team of people can use Designer/2000 to effectively and efficiently share the design effort. Team-based

design tools make it easier to share work and ideas and reduce the dependency on one particular team member.

Designer/2000 is not *a* tool; it is a set of tools that interface and work together for the same goal. The Designer/2000 suite of tools includes the following products:

- ◆ **Process Modeller.** This tool is used to assist in modeling your system based on your business process.
- ◆ **Systems Modeller.** This tool is used to assist in modeling your system based on the information available in your system.
- ◆ **Systems Designer.** This is the tool that supports the design of the information system.
- ◆ **Generators.** These utilities take the data models you have created with the other Designer/2000 tools and create client-side and server-side code to help in the creation of data definitions and applications.

The following sections look briefly at each of these design tools.

Process Modeller

The Process Modeller focuses on the essentials of the process for which the information system is designed. These essentials include the primary steps of the business process, how these processes are linked, and who is responsible for these steps. These elements are the fundamentals of the business process, including how the organization operates. If the business process is not correctly modeled, it is impossible to effectively design the information system to support the business process.

Systems Modeller

Another way to base your system design is on the information itself rather than on the process. Either of these methods is effective. The Systems Modeller takes the information used in the business process as the basis for the system's design.

The Systems Modeller utility includes the following components:

- ◆ Entity Relationship Diagrammer, used to construct entity relationship models.
- ◆ Function Hierarchy Diagrammer, used for the construction of functional hierarchies.
- ◆ Dataflow Diagrammer, used for the construction and testing of dataflow models.

These components are used to diagram and relate the information used by various organizations within your corporation.

Systems Designer

The Systems Designer is the component that actually takes the process and relationship information and supports you in your system-design tasks. The output of the System Designer tools includes such things as these:

- ◆ Data definitions
- ◆ Business rules
- ◆ Procedure definitions
- ◆ Transaction definitions
- ◆ Data usage definitions
- ◆ Workflow definitions
- ◆ Queries and reports

The System Designer allows you to input and output additional information concerning the structure of your system and process rules. The System Designer includes tools such as a Data Diagrammer and a Structure Diagrammer that allow the design of individual modules. Information from your data models is used to verify that relationships and process rules are followed. The System Designer includes many design wizards to assist in the creation of these modules.

Generators

Generators are the tools that take the models designed by the System Designer and produce client and server code to help you create your database and application. The generators take into account not only the models but your business rules as well to help design an application that fits your needs. The generators include the following tools:

- ◆ **Forms Generator.** This tool designs Oracle forms based on the business information required.
- ◆ **Reports Generator.** This tool designs Oracle reports based on the business reports your organization requires.
- ◆ **Server Generator.** This tool creates the server code necessary for the management of objects, tables, views, and so on.

These tools make Oracle Designer/2000 a very powerful package. Designer/2000 is a powerful and complex combination of tools that can help with the entire design process of your information system. Because it is quite complex, it is not for smaller installations that have only a few tables with relatively few relationships.

Third-Party Tools

Several very good utilities that can assist you with the design of your database are on the market today. Among others, these third-party tools include ERwin/ERX from Logic Works, S-Designor from SDP Technologies, and SQL Coder from Platinum Technologies. All three of these tools are very good and easy to use; all three are on a much smaller scale than the Designer/2000 suite from Oracle.

If you are in a small-to-medium business and don't have the need for a tool as complex and sophisticated as Designer/2000, one of these third-party products may be just for you.

NOTE: The three tools described in the following sections are used as examples of what is currently available; do not consider the inclusion or exclusion of other similar tools in this book as an endorsement or lack of endorsement of any tool.

ERwin/ERX from Logic Works

ERwin/ERX is similar to Designer/2000 but is not as complex to use. ERwin/ERX graphically builds relationships between tables; from that information, it produces Oracle definition code that can assist you in building the database. ERwin/ERX can also create PL/SQL code for triggers and stored procedures.

By using a graphical tool such as ERwin/ERX, it is easy to view the relationships between tables and clusters and see how constraints must function. The use of such a tool can help you visually represent the relationships between tables you might otherwise miss when designing a database by hand.

S-Designor from SDP Technologies

S-Designor is a third-party tool that functions in much the same way as Designer/2000 but on a smaller scale. S-Designor is a Windows application that graphically lets you input your business and application models; from that information, it creates Oracle definition code you can use to create an Oracle database.

S-Designor interfaces with tools such as PowerBuilder, SQLWindows, and others to provide a complete database and application design tool. With S-Designor, you can create not only database definitions but constraints, views, triggers, and stored procedures as well.

Because S-Designor is not as sophisticated and complex as Designer/2000, it may be ideal for moderate-to-small database installations. Because the product works with application design tools with which you are already familiar, no additional overhead is incurred by having to learn a new application development tool.

SQL Coder from Platinum Technologies

SQL Coder is another excellent tool that can help you properly design and implement your server database. SQL Coder can assist you to properly implement objects such as stored procedures, triggers, packages, views, tables, and so on. By employing this kind of tool to assist you in the design phase, you can avoid errors and design a more optimal database.

By examining the layout of your database, SQL Coder can help with the design of the database and can offer suggestions, such as where indexes would be useful. As with the other database design tools mentioned in this chapter, SQL Coder can be used standalone or in conjunction with application development tools available from Platinum Technologies or other vendors.

Summary of Database Design Tools

The tools described in the preceding sections—as well as other third-party tools—can assist you in the design of your database and help you determine relationships and constraints you might otherwise miss when creating the database by hand. When dealing with tens or hundreds of tables, it can be difficult to coordinate all the relationships and constraints.

When you use tools to assist with the design stage of the database, you can avoid costly mistakes. By using tools, you simplify your work so that you can focus your attention on the design process and database design rather than on all the details. No matter how good the tool, remember that it is only a tool; all database information should be verified before you build your database.

Application Development Tools

You can use application development tools to assist in the prototyping, development, and deployment of applications. When you use application development tools, you can quickly deploy very good applications with relatively little effort. However, you should keep in mind a few caveats when using this type of tool.

Some tools are easy to use and require very little training before applications can be deployed. However, the more sophisticated tools typically allow you to create much more sophisticated applications. This is not to say that the easy-to-use application development tools are not good—they can be very good. But some of them may have some limitations if you try to use them for very sophisticated applications.

Oracle Tools

In addition to its database design tools, Oracle has several development tools to offer. Oracle's Power Objects is similar to the third-party tools described later in this chapter. Power Objects is a graphical "drag-and-drop" tool that includes a small database called BLAZE. You can use this small standalone RDBMS to assist in the development of applications and also for small standalone applications. Power Objects' real power comes when you use it as an application development tool. Power Objects is currently available on the Windows and Macintosh platforms.

Oracle also offers the traditional Cooperative Development Environment (CDE) tools that have evolved into the Oracle Developer/2000 development tools. These tools have been in production for quite some time on almost all the platforms Oracle supports. Developer/2000 added flexibility that allows you to develop applications across a variety of platforms using the same base code set. The Oracle Developer/2000 tools are described in detail in Chapter 35, "Using GUI Builders," along with how to tune the applications developed by these tools.

Oracle Power Objects

Oracle Power Objects is a standalone, object-oriented development tool you can use to develop client/server applications or standalone applications by using the small BLAZE database provided with it. Oracle Power Objects allows you to drag-and-drop everything. In this manner, you can quickly develop and test small applications. Although Power Objects is great for developing small standalone applications, its real power comes from its use as an application development tool. Power Objects is currently available for Windows and Macintosh.

Oracle Developer/2000

Oracle Developer/2000 is a higher-end tool than Power Objects. Oracle Developer/2000 tools are the follow-on to the traditional CDE tools offered by Oracle. The Developer/2000 tools allow you to design graphical applications and then simply compile them to run in a Windows, Macintosh, Motif, or character-based mode. This flexibility allows standardized applications to be deployed across a heterogeneous network of different machines.

Third-Party Tools

Third-party tools you can use to facilitate application development include products such as PowerBuilder from PowerSoft, SQLWindows from Gupta, and Delphi from Borland. Each of these tools is very good and can help speed the development of graphical applications in a Windows or Macintosh environment. These tools are all graphical in nature and allow you to build applications from a graphical console in a drag-and-drop fashion.

PowerBuilder from PowerSoft

PowerBuilder from PowerSoft is by far one of the most popular application development tools. PowerSoft was among the first to introduce these types of products. PowerBuilder comes with ODBC drivers and native drivers that allow you to connect to SQL*Net directly. Using PowerBuilder, you can very quickly prototype and test various applications.

SQLWindows from Gupta

Gupta's SQLWindows is another very popular application development tool. SQLWindows uses a wizard called QuickForms to help in the development process. Using this tool, you can easily and quickly develop a usable application. SQLWindows comes with ODBC drivers and recommends using them as the primary method of connection. The SQLWindows forms tool is very easy to use and can quickly have you building and running applications.

Delphi from Borland

Delphi is a graphical, drag-and-drop tool that is very easy to use and very powerful. Delphi comes with ODBC drivers and an optional SQL*Net driver. Delphi consists of two main components: the forms developer is simply called *Delphi*; the reports developer is called *ReportSmith*.

These tools are very easy to learn; you can be up and building applications in just a few hours. These tools connect to your database and extract the data types directly from Oracle without requiring manual insertion of these values. If you have to deploy applications in a short time frame, I recommend that you use these tools. More detailed descriptions of these tools and ideas for more efficiently tuning the resulting application are given in Chapter 35, “GUI Builders.”

Analysis Tools

Several analysis tools available from Oracle and third parties are useful in tracking down inefficiencies and bottlenecks in SQL statements. These tools work both on the client machines and on the server itself. These analysis tools can assist you in tracking down a particular SQL statement that is causing problems or can point to a particular table or cluster that is inefficient.

Two of the Oracle tools—Oracle Expert and Oracle Trace—are part of the new Oracle Mission Control console. Oracle Mission Control is a graphical console designed to assist you in monitoring, managing, and tuning your Oracle systems.

Oracle Expert is designed to look for and correct system bottlenecks. Oracle has recently introduced the Oracle Trace product, which originated with Digital RDB. Oracle Trace can be used in a manner similar to SQL Trace and can be extended to the network and applications.

Third-party tools are also available to analyze SQL statements and performance; these include tools from Platinum Technologies, Precise Software Solutions, Compuware, Eventus, and BMC. The following sections look at some of the analysis tools currently available.

Oracle Mission Control

Oracle’s Mission Control console is a graphical management station that includes some of the following features:

- ◆ **Monitoring and Alerting.** This feature allows for threshold monitoring and event management, which can consist of paging a DBA or performing an automatic corrective action.
- ◆ **Automated Task Scheduling.** Automated maintenance tasks can be scheduled and run during off-hours, unattended.
- ◆ **Monitoring and Diagnostics.** This feature can allow the database administrator to trace connections and diagnose problems in real time.
- ◆ **SNMP Monitoring.** The Oracle Mission Control console provides SNMP information to third-party management consoles.
- ◆ **Tuning.** When you use Oracle Expert, tuning can be assisted by the expert system.
- ◆ **Whole System Monitoring.** This feature allows you to monitor your entire environment from a single console.

Oracle Trace

Oracle Trace is a new product based on technology acquired when Oracle purchased the RDB product from Digital Equipment Corporation. Oracle Trace takes SQL Trace to client/server computing.

NOTE: *Oracle Trace* is the product name for the new client/server trace facility. Although it is similar in function, it should not be confused with SQL Trace.

Oracle Trace begins where SQL Trace left off. The key feature of Oracle Trace is its ability to monitor applications and third-party tools using the SQL Trace function. Oracle Trace collects performance data on predefined events in the Oracle Server, SQL*Net, Oracle Forms, or any third-party application that has been written to take advantage of Oracle Trace.

Oracle Trace provides information similar to the data you expect from SQL Trace:

- ◆ CPU time
- ◆ Memory usage
- ◆ Page faults
- ◆ Other user-defined data

By allowing you to define your own data to monitor—as either an end user or an applications developer—this tool offers tremendous flexibility. Oracle Trace information can be viewed with the Oracle Mission Control console and will soon be available in third-party performance-monitoring tools. By tracing your application from the client, through the network, and to the server, performance bottlenecks can more easily be detected and removed.

Third-Party Tools

A number of fine products are on the market today, designed to help you analyze your system, determine bottlenecks, and remove those bottlenecks. Some of these tools are general; others focus on a specific area. The following sections take a sampling of the various tools that serve different purposes.

NOTE: Many tools are available on the market; the fact that I have chosen just a few is not an endorsement of a particular product. Because products are constantly being introduced and improved, I have decided to focus on the products with which I am most familiar. Because there are many great products I have not mentioned, I suggest that you look at the market before purchasing an analysis tool.

TSreorg from Platinum Technologies

TSreorg is a graphical tool designed specifically to monitor the structure of your database and to reorganize and defragment the objects in your database to maximize I/O performance. Although TSreorg performs other functions such as job monitoring and scheduling, the real beauty of this product is its ability to reorganize and defragment. Platinum Technologies also offers an excellent suite of system-management products.

Precise/SQL Suite from Precise Software Solutions

Precise Software Solutions offer two tools to help analyze problems within your applications:

- ◆ **Inspect/SQL.** Used to identify problems within applications themselves.
- ◆ **Analyze/SQL.** Used to analyze problems and model potential improvements.

This product is interesting because it focuses on the client application, which is where many performance problems originate.

DATA-XPERT from Compuware

DATA-XPERT is a useful tool for testing a database that has not yet gone into production. DATA-XPERT is designed to create sample data you can use to test your database. By using a tool like this to generate good quality test data, you can find and fix potential problems before your database and applications are deployed.

AdHawk from Eventus

Although AdHawk is usually considered a database monitoring tool, it can offer specific recommendations for resolving database problems that may occur. Eventus also offers AdHawk Spacer, a tool that provides for database reorganization and management.

DBGENERAL from Bradmark

Although Bradmark's DBGENERAL is also primarily seen as a database monitoring tool, the most recent release has increased its functionality. DBGENERAL now provides for reorganization, defragmentation, and capacity management as well as object management.

Patrol from BMC

Patrol is a powerful tool for monitoring and alerting based on events. Patrol uses knowledge modules to monitor certain events and to communicate with the Patrol console. You can use these knowledge modules to create your own events within the application itself. In this manner, you can monitor every aspect of your system.

Summary of Analysis Tools

As you have seen in the preceding sections, a number of excellent products are on the market to help with various aspects of managing your system and detecting and removing performance bottlenecks. When you take advantage of these tools, your system can continue to perform optimally after deployment.

Remember that the job of optimizing and tuning your system does not stop once the system goes into production. It is an ongoing job to keep your machine tuned and running well at all times. The best way to solve performance problems in production systems is to identify and fix any problems before they start affecting users. In this way, you can avoid costly downtime and loss of work.

Summary

By taking advantage of SQL development tools for the design of your database, the development of your applications, and for analyzing the outcome of your applications, you can facilitate the optimization of your system. Database design tools can be used to determine where the use of clusters or indexes can help the efficiency of the system. Application development tools can help you rapidly develop, prototype, and deploy graphical applications with a minimal amount of effort.

Other tools available on the market can perform specific functions to help remove some of the bottlenecks they discover. Some of these tools perform defragmentation functions; others re-organize database objects for optimal I/O performance. The use of such tools to keep your system running optimally outweighs the purchase price of these products. Performance tuning and optimization does not stop when the system is deployed; it is a ongoing job that can be assisted by the use of the right tool.

The use of SQL analysis and tracing tools can assist you in tracking down problem areas and bottlenecks in both the database server and the application. This capability can be useful not only in indicating the cause of a performance problem or bottleneck but also in ruling out areas in question. Performance problems can be associated with the client, the server, or a combination of the two. SQL tracing tools facilitate the determination of the problem and may lead to a solution.

Chapter 32

Miscellaneous SQL Topics

To wrap up Part IV of this book, “Tuning SQL,” this chapter covers some miscellaneous topics. This chapter is mainly a collection of unrelated tips on ways to improve SQL statements in certain areas.

Here are some of the tips covered in this chapter:

- ◆ **Table sequences.** This section of the chapter looks at the Oracle sequence generator and how to take advantage of cached sequences for primary key values.
- ◆ **Join performance.** This section describes how best to optimize the performance of different join operations.
- ◆ **Locking.** This section describes locking and how best to take advantage of Oracle's locking features.
- ◆ **Array processing.** This section looks at how to take advantage of the Oracle array processing feature.

Although these topics are not large enough to warrant their own chapters, they are no less important than the other SQL tuning topics described in Part IV of this book.

Table Sequences

It is frequently necessary to generate a sequence of numbers to use in your database. You may need these numbers to identify a particular record or for some other purpose. To create a unique sequence of numbers on your own, you would have to lock the record that has the last value of the sequence, generate a new value, and then unlock the record. To avoid locking these records, Oracle provides a sequence generator that performs this service for you.

The Oracle sequence generator can generate sequential numbers of up to 38 digits, without having to manually lock records. When you define a sequence, you can specify the original values of the sequence, whether the sequence should be cached, and whether the sequence should be ascending or descending.

Creating Sequences

Sequences are created with the `CREATE SEQUENCE` command. This command has the following syntax:

```
CREATE SEQUENCE dog_breeds
  INCREMENT BY 1
  START WITH 1
  NOMAXVALUE
  NOCYCLE
  CACHE 4;
```

The following chart provides brief descriptions of the parameters.

Parameter	Description
INCREMENT BY	Specifies the amount to increment the sequence by each time a value is obtained.
START WITH	Specifies the starting value.

<i>Parameter</i>	<i>Description</i>
MAXVALUE <i>n</i>	Specifies the maximum value that the sequence can obtain.
NOMAXVALUE (Default)	Specifies no maximum value for a sequence. The sequence can grow to 10^{27} for ascending sequences and -1 for descending sequences.
MINVALUE <i>n</i>	Specifies the minimum value of the sequence.
NOMINVALUE (Default)	Specifies no minimum value for a sequence. The sequence can have a minimum of 1 for ascending sequences and -10^{26} for descending sequences.
CYCLE	Specifies that a sequence will restart after reaching the maximum or minimum value.
NOCYCLE (Default)	Specifies that the sequence cannot recycle after reaching the maximum or minimum value.
CACHE <i>n</i>	Specifies the number of sequence entries to cache for quick access. (The default is 20 values.)
NOCACHE	Specifies that no sequence entries should be cached.
ORDER	Specifies that sequence entries are generated in the order in which they are requested. By default, this is not the case.
NOORDER (Default)	Specifies that sequence numbers are not necessarily generated in the order in which they are requested. This is usually fine for primary key values.

Tuning Sequences

To get the best performance out of sequences, you should cache as many sequences as you think you will have simultaneous requests for them. By over-specifying the number of cached sequences, you use more memory than necessary. By under-specifying the number of cached entries, you cause undo waiting for the sequences.

The Oracle sequence generator is much more efficient than any way you can manually generate sequences. If you have a series of values that must be sequential, I recommend using the Oracle sequence generator.

Using Sequences

To generate a new sequence value, simply reference the value of `sequence_name.NEXTVAL`. To re-reference that number from within the same SQL block, simply reference the value of

`sequence_name.CURVAL`. When you reference `sequence_name.NEXTVAL`, a new sequence number is generated.

Here is an example using the sequence created in the preceding section to generate a new value in the BREEDS table. The result of this `INSERT` statement is to insert a breed number that has a sequentially growing value in the BREED column. (Remember that the BREED column of the BREEDS table is a sequential value for the dog breed. This value is referenced by the BREED column in the DOGS table.)

```
SQL> INSERT INTO breeds
  2  ( breed, breed_name, description )
  3  VALUES
  4  ( dog_breeds.NEXTVAL, 'Australian Cattle Dog', 'Hard working and tough' );
1 row created.
```

Sequences are incremented as they are accessed, independent of rollback or commit. If a transaction generates a sequence and then rolls back, the sequence is not replaced. Therefore, there may be holes in your sequential values. This is usually not a problem.

NOTE: Because sequences are generated independently of commits or rollbacks, you may have gaps in the sequences. Although this is usually not an issue, you should make a note of it.

Using Cached Sequences for Primary Key Values

As shown in the preceding example, it can be efficient to use cached sequences to generate unique primary-key values. Not only is the performance of the cached sequence good, you are guaranteed a unique number (unless you have enabled `CYCLE`).

CAUTION: If you use cached sequences to generate primary-key values, be sure to set the `NOCYCLE` parameter for the sequence and make sure that the minimum and maximum values are sufficiently high. Cycling sequences causes integrity constraints to be violated.

Review of Sequences

Now you know how to create sequences and use them to generate unique values that can be used as primary keys. The Oracle sequence generator is a powerful tool that can be quite useful when needed. I recommend that you use the Oracle sequence generator to generate any sequential primary-key value.

By taking advantage of this feature, you can simplify application coding and improve performance. Performance is improved by reducing the manual locking and additional SQL statements that would be required to generate these unique values on your own. Although the sequence generator on its own may not be a huge performance improvement, in conjunction with many other optimizations, overall system performance benefits.

Join Performance

A join operation occurs when rows from two or more tables are combined in a single query. Typically, joins occur in the `WHERE` clause of SQL statements. The condition for the select in the `WHERE` clause is known as the *join condition*. There are several different types of joins, as shown in the following chart:

<i>Join Type</i>	<i>Description</i>
Equijoin	A join in which the pairs are joined with an equality condition. For example, the statement <code>SELECT dogs.dogname, breeds.description FROM dogs, breeds where dogs.breed = breeds.breed;</code> is an equijoin; it returns only those rows in which the value of <code>dogs.breed</code> is equal to <code>breeds.breed</code> .
Self join	A join in which a table is joined to itself. The self join is similar to the equijoin except that the same table is used for both components.
Cartesian product	The product of a join with no join condition. The result contains many rows. Unless you specifically need a Cartesian product, avoid this type of join. An example of a Cartesian product is the result of the <code>SELECT * from dogs, breeds;</code> query. A Cartesian product is usually the result of a mistake in a join statement and should be avoided.
Outer joins	The result of an outer join is the rows that satisfy the join condition AND those rows in the first table for which no rows in the second table satisfy the join condition. For example, the statement <code>SELECT dogs.dogname, breeds.description FROM dogs, breeds where dogs.breed = breeds.breed (+);</code> is an outer join and returns only those rows in which the value of <code>dogs.breed</code> is equal to <code>breeds.breed</code> and all other rows in which the join condition is not met.

The following sections describe each of these types of joins, their tuning considerations, and ways of improving performance.

Equijoin

Equijoins occur when an equality operator is specified in the join condition. You can improve the performance of equijoins by specifying a join condition that can take advantage of indexes. If this join condition is frequently used, you have several options:

- ◆ Create indexes on the join condition columns. If indexes do not already exist, it will improve performance if the values of those columns are somewhat unique.
- ◆ Modify the join condition. It may be necessary to modify the join condition to take advantage of already-existing indexes.
- ◆ Create a cluster. If these tables meet the conditions of a cluster (as described in Chapter 26, “Tuning SQL Statements”), you will see significant benefits from clustering.

The benefit you see from using these options depends on the data and the application. If a majority of your operations on these tables are joins, you should seriously consider each of these options.

Self Join

A *selfjoin* occurs when the `WHERE` clause in a `SELECT` statement references another column in the same table. The performance of a self join can be improved by specifying a join condition that takes advantage of indexes. If this join condition is frequently used, you have options similar to those available for the equijoin:

- ◆ Create indexes on the join condition columns. If indexes do not already exist, it will improve performance if the values of those columns are somewhat unique.
- ◆ Modify the join condition. It may be necessary to modify the join condition to take advantage of already-existing indexes.

Note that creating a cluster for a self join is not an option.

Cartesian Product

A *Cartesian product* is the product of a join with no join condition; the result contains many rows. Unless you specifically need a Cartesian product, avoid this type of join. In my experience, Cartesian products are rarely of much value. The best way to improve performance for a Cartesian product is to avoid using one.

Outer Join

The result of an *outer join* is the rows that satisfy the join condition AND those rows in the first table for which no rows in the second table satisfy the join condition. As with the equijoin operation, you can improve performance for an outer join by specifying a join condition that takes advantage of indexes. If this join condition is frequently used, you have several options:

- ◆ Create indexes on the join condition columns. If indexes do not already exist, it will improve performance if the values of those columns are somewhat unique.
- ◆ Modify the join condition. You may have to modify the join condition to take advantage of already-existing indexes.
- ◆ Create a cluster. If these tables meet the conditions of a cluster (as described in Chapter 26, “Tuning SQL Statements”), you can see significant benefits from clustering.

The benefit of using these options with an outer join depends on the data and the application. If a majority of your operations on these tables are joins, you should seriously consider each of these options.

Tuning Joins for Throughput

To specify that the join operation be optimized for maximum throughput, specify the `USE_MERGE` hint to force the optimizer to choose a sort-merge join. Most likely, this is the optimization approach the optimizer would choose by default anyway, unless another approach was defined with the Oracle initialization parameters.

Tuning Joins for Response Time

To specify that the join operation be optimized for response time, specify the `USE_NL` hint to force the optimizer to choose a nested-loops join. Remember that with the `USE_NL` hint, you must specify the inner table in the nested loop. On its own, the optimizer would not choose this approach.

Review of Joins

Join operations are quite often used in applications and are a necessary part of doing business. By taking advantage of Oracle’s index and cluster features, you can improve the performance of these join operations. By understanding your data and your application, you can determine whether the database can benefit from these features. Understanding your data and your application is the key.

Locking

A book on Oracle performance is not complete without a discussion about locking. With Oracle version 6, the concept of *row-level locking* was introduced. Although row-level locking improves performance, there are conditions in which table-level locking is necessary.

What Is Locking?

In multiuser systems or servers, many users may update the same information at the same time. Locking allows this to occur and ensures that data you are currently modifying cannot be modified at the same time by a different user.

The basic idea of locking is that when a user accesses data through a transaction, that data is *locked* (that is, it is exclusively owned by that transaction) until the transaction is committed or rolled back. Because data cannot be accessed by other transactions until the lock is released, it is to the benefit of the system that you reduce both the time the data is locked and the amount of data that is locked.

Oracle provides two basic types of locks: the row-level lock and the table-level lock.

When you use a **row-level lock**, only the row in the table that is being modified is locked. Other users can access data and update other rows in that table; they just can't update the particular row being modified.

All rows in the table can be viewed—even the one being modified. Other users who view the row being modified see the old version of the data (through the rollback segment) until the changes are committed.

Row-level locking is more efficient than table-level locking because it does not keep other transactions from running. OLTP systems benefit from row-level locking because of the large number of concurrent users who access the system.

When you use a **table-level lock**, the entire table being modified is locked. Other users can't modify any rows in the table until the lock is released.

As with the row-level lock, other users can still view data being modified; they see the old version of the data (through the rollback segments) until the changes are committed.

Serializable Reads

If your application requires serializable reads, you must change the parameters `SERIALIZABLE` and `ROW_LOCKING` in the Oracle initialization file. By changing these parameters, you alter the performance of the system dramatically. Oracle recommends that users implement applications using the default row locking.

When you set the Oracle initialization parameter `SERIALIZABLE` to TRUE, queries acquire table-level read locks, thus preventing updates to the objects being read until the transaction containing the query has been committed. This mode provides for repeatable reads and ensures that two queries for the same data in the same transaction see the same data.

This parameter provides for ANSI-degree-three consistency—but at a considerable cost in concurrency. I recommend that you do **not** set `SERIALIZABLE` unless it is absolutely necessary to correctly run older applications. In most cases, row-level locking is optimal.

Using Locks

Locks are usually handled transparently for the user and are created and released automatically when the transaction has been committed or rolled back. Even with this transparency, it is important for you to realize that the locks are held until the commit or rollback. Long-running ad-hoc transactions that are not committed can cause contention problems on the system if they are not addressed.

CAUTION: Transactions that have not been committed or rolled back continue to hold locks. Don't forget to commit or roll back your transactions.

Explicit Locking

Tables can be explicitly locked by using the `LOCK TABLE` command. With this command, you can manually lock and unlock tables in different modes. By manually locking tables, you can have a lot of flexibility over how the locks are done—but there is a danger.

Locking tables manually can result in locks that are held longer than necessary and a loss of flexibility for the application. Manual locking should be reserved for the most-experienced programmer and only under extreme conditions. I don't recommend using manual locking at all.

The `SELECT...FOR UPDATE` Statement

The `SELECT...FOR UPDATE` statement allows you to explicitly lock a set of rows that you will be updating. This statement allows you to update several rows as a group and be assured that none of the rows can change until you have finished your transaction. In this manner, no other process can acquire a lock on one of the elements in your transaction until you finish with all of them. Using the `SELECT...FOR UPDATE` statement is much more efficient than locking the table explicitly.

Deadlocks

Deadlocks occur when two (or more) processes hold resources that the other one needs. Because neither of the processes will release its resource until it has received the other's resource, neither can proceed. If a deadlock occurs, Oracle breaks the deadlock by forcing one or more of the processes to roll back and release the resources. By carefully designing your application to acquire locks in sequences, you can avoid deadlocks.

Review of Locking

Locking is an important part of a multiuser RDBMS system. Relying on Oracle's internal processing to handle locking for you is usually quite efficient. Unless absolutely necessary, take advantage of Oracle's row-locking features and avoid table locks. By relying on the internal features of Oracle, you can simplify application development. Of course, more sophisticated applications may have to perform manual-locking operations, but don't do so unless absolutely necessary.

Array Processing

When inserting a large number of rows into a table, you may be able to improve performance by taking advantage of *array processing*. Array processing allows multiple values to be passed to Oracle in one statement. Array processing reduces the number of calls to Oracle and reduces network overhead because a large network packet is much more efficient to send than many small packets.

Consider the following embedded SQL statement:

```
INSERT INTO dogs
( dogname, age, breed, owner )
VALUES
( :d_name, :d_age, :d_breed, :d_owner );
```

If the variables bound to `d_name`, `d_age`, `d_breed`, and `d_owner` are simple variables, then Oracle executes one statement. If the variables bound to those names are arrays, the entire array is sent to Oracle for processing. The SQL loader takes advantage of array processing to load rows in the most efficient manner possible; so do the Import and Export utilities.

Using VARCHAR2 instead of CHAR

By using the VARCHAR2 data type instead of the CHAR data type, you can eliminate unnecessary blank padding. The VARCHAR2 variable type stores a variable-length string; CHAR has a fixed-length format. By using VARCHAR2 instead of CHAR, you save space in the database as well as network and CPU resources. Although the VARCHAR data type is also supported, its use is not recommended.

Summary

This chapter is intended to wrap up all the miscellaneous tips and suggestions concerning the tuning of SQL statements. The chapter was a collection of unrelated items that didn't quite fit in any of the other chapters in Part IV of this book. Having finished this chapter, you should be familiar with the following topics:

- ◆ Sequences and when they are useful
- ◆ Join operations and how to improve their performance
- ◆ Array processing and how to take advantage of it
- ◆ Oracle locking methods and the difference between row-level locking and table-level locking
- ◆ The difference between the VARCHAR2 and CHAR data types

Having finished Part IV of this book, “Tuning SQL,” you should have a solid understanding of the importance of tuning your SQL statements and a better idea of how to do that.

Part



Tuning the Client

Chapter 33 What Affects Client Performance?

- 34** Tuning the Client System
- 35** Using GUI Builders
- 36** Using Middleware Products

In Part IV of this book, you saw how poorly tuned SQL statements can cause inefficient processing and optimization. You learned how to determine whether your SQL statements are performing unnecessary functions and causing unwanted data to be returned to your client system. You learned how to use Oracle tools such as SQL Trace and EXPLAIN PLAN to determine whether your SQL statements are inefficient. You also looked at how to take advantage of the Oracle optimizer and how to use procedures and packages to best utilize the shared SQL area of the shared pool. Finally, you looked at some products available to help you debug and optimize SQL statements.

Part V complements Part IV by looking at specific ways to tune the client system. The chapters begin by looking at what a client system is and how two-tiered architectures differ from three-tiered architectures. You learn that a “client machine” may be more than you are used to thinking it is and discover how to take advantage of the power of the client machine.

You will learn to determine whether a problem exists within the client machine itself and how to solve the problem if you find one. You will look at what affects the performance of the client system and how to reduce bottlenecks. You also examine some specific client operating systems such as Microsoft Windows, Microsoft Windows NT, and UNIX.

Several GUI (Graphical User Interface) application design tools (such as PowerBuilder, SQLWindows, and Oracle Power Objects) are introduced; you determine ways to optimize the applications you can build with these tools. Finally, you look at some other options for optimizing the database clients, such as the use of a Transaction Monitor (TM).

When you finish this part of the book, you should be able to determine whether the client is causing a bottleneck in the system and how you can tune the client. You should also be able to take applications built by GUI application builders and tune them for optimal performance. If you tune the client, the SQL statements, and the server for optimal performance, you will have a finely tuned system that has been optimized for your particular configuration and application.

Chapter 33

What Affects Client Performance?

This chapter examines the things that affect the performance of the client machine and explains how to optimize the client. During the last few years, the definition of what a client machine is has changed. The first thing this chapter does is define what is meant by the *client*.

Originally, application development tools did nothing more than create queries and display them. The tools have changed for the better. Now, these tools create applications that can take advantage of the power of the client, offload CPU processing from the server, and add functionality such as video processing, compression, and presentation services.

Once you define what the client is, you can start to see ways you can optimize performance. The *client system*, in the traditional sense, is changing. What we have traditionally called the client is evolving into a sophisticated database query-and-presentation tool.

What Is a Client Machine?

To define what a *client* is, it may help to look at the evolution of computing over the last few years. By looking at where we came from and where we are today, it may be easier to see where we will be going in the future.

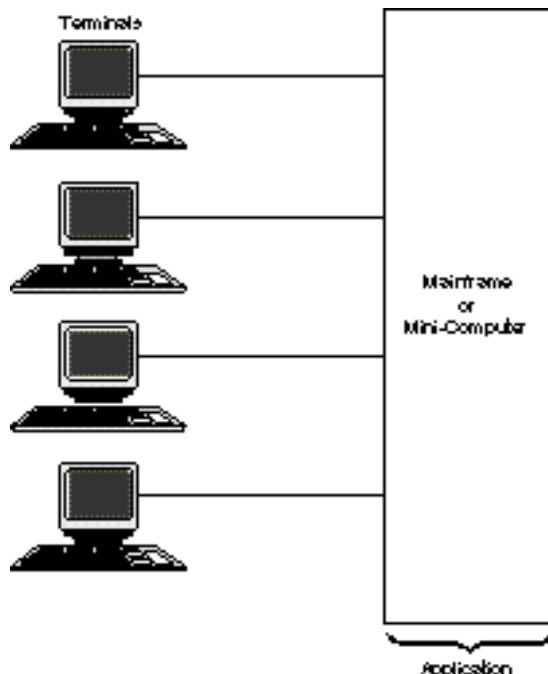
The following sections start out by examining the *traditional computing model*, which began with the development of computing as we know it. Following that is a discussion of the emergence of network computing, which led to what I call *GUI/server computing*. This evolution led to what we know today as *client/server computing*. (The way I define *client/server computing* may be slightly different from the way you traditionally see it.)

The Traditional Computing Model

When computers were first developed, the only way to input data was with keypunch cards. This approach was extremely painful and the industry quickly evolved to timesharing systems and the use of terminals (see Figure 33.1). The terminal model was quite effective and dominated the industry for quite some time; it is still an extremely popular way to do computing.

Figure 33.1

The traditional, terminal-based computing model.



Terminal-based computing saw the introduction of many innovations that allowed for ease of use in applications:

- ◆ **Field validation.** This allowed the application itself to determine whether the correct field type was input. Types such as numeric, character, and so on were validated.
- ◆ **Screen positioning.** Innovations such as the ability to tab between fields and move forward and backward were added to improve usability.
- ◆ **Menu driven.** Many early character-based applications allowed menus of choices to be displayed.
- ◆ **Screen presentation.** Probably the best innovation introduced by terminal-based applications was the ability to format the presentation of data. This included tabular formats that allowed scrolling.

These innovations really improved the usability of applications and allowed difficult tasks to be simplified. The main benefit of terminal-based applications was their ability to run on a number of different platforms and terminals. Even today, it is sometimes difficult to run an application built for one type of computer or operating system on a different type of computer.

Many of these innovations have been carried forward into what most of us view as “more modern applications”: the kinds of applications we are used to seeing today.

The Network Computing Model

The next step in the evolution of computing came when networks became a large part of the industry. Many applications stayed very much as they were with terminal-based computing, but allowed access through a computer network (see Figure 33.2). Although these applications allowed remote users to log in to the host computer and run the application, the application itself was still terminal or character based.

The users accessing the application were connected through a terminal to a host machine (or perhaps to a workstation or PC on the network running DOS or Windows). In this step of the evolutionary process, the user was separated from the direct connect. The application remained unchanged and no major improvements occurred.

The GUI/Server Model

The next major step in the evolution of computing came when application vendors started to develop Graphical User Interfaces (GUIs) for their applications (see Figure 33.3). Although some people call this kind of computing *client/server*, I do not. I think that for a system to be considered a *client*, it must have more than just presentation services.

Many of these GUI/server applications do nothing more than their terminal counterparts did—they just added a pretty border or the ability to pull down menus with the mouse rather than with a keystroke. In fact, many graphical applications have a terminal-based or character-based version that has the exact same functionality.

Figure 33.2

The network computing model.

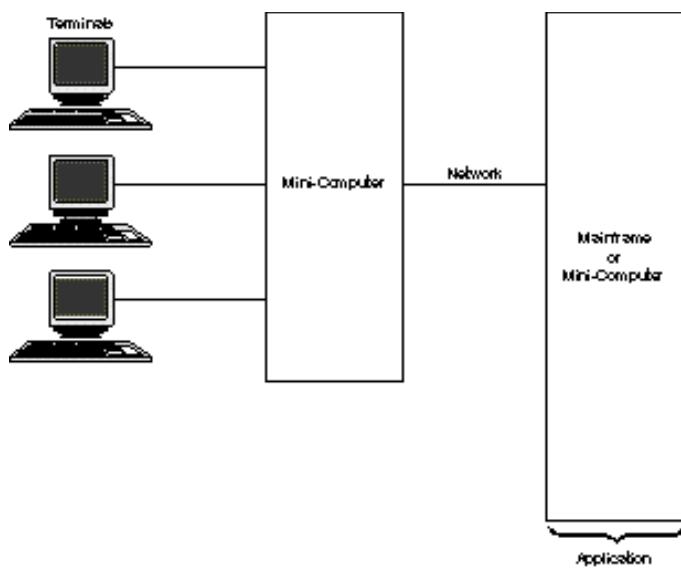
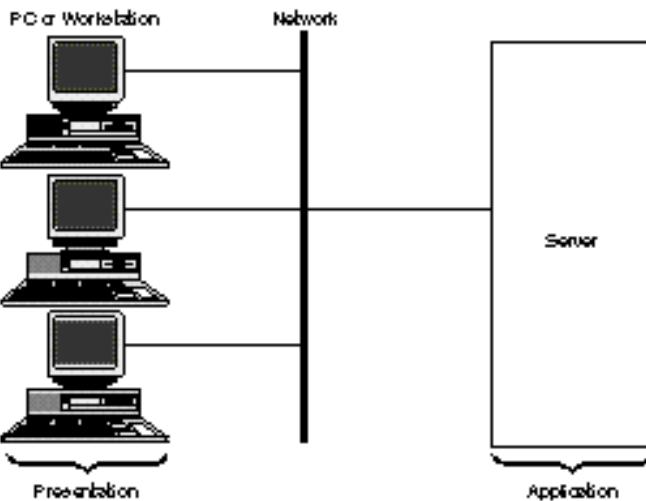


Figure 33.3

The GUI/server computing model.



Many early GUI development tools were notorious for building a pretty screen but generating horrible SQL code. The SQL statements they generated were very inefficient and often generated much unnecessary CPU and network processing. Much of the problem was caused by the fact that the development tool generated code that was interpreted by another application that ran on the client. This arrangement was very inefficient and slow. It was not uncommon for complaints from users to result in the discovery that a 10-second response time was caused by 2 seconds in the RDBMS and 8 seconds in the GUI. Fortunately, today's tools are much more efficient.

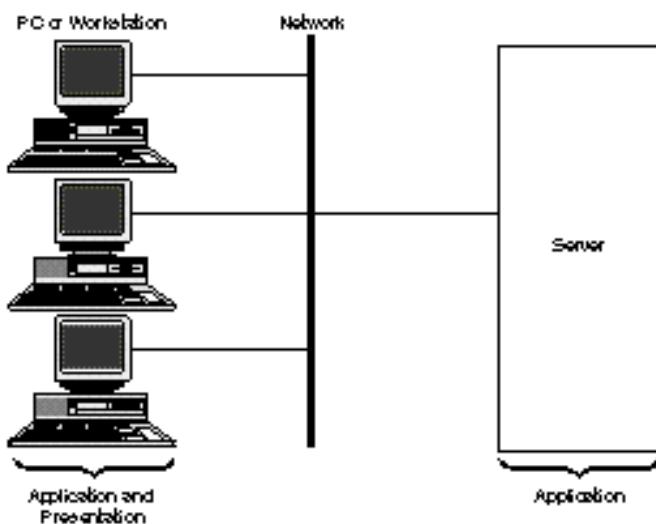
The addition of the GUI to applications changed the way people looked at applications but was not a great leap in technology. The real revolution occurred when these “client” machines started to do more than just present the data to the user.

The Client/Server Model

The client/server came of age when the “client” actually started doing work. Allowing the client to offload work from the server does help somewhat, but the real benefit of client/server computing is the addition of new functionality that was impossible in the past. The client/server machine allows many new and traditional functions to be performed on the client machine (see Figure 33.4).

Figure 33.4

The client/server computing model.



In general, these client functions have very little to do with the actual display of the data (in contrast to the GUI/server model, which affected only the display of data). These additional functions have more to do with processing the data to be displayed, validating input values, performing local table lookups, and so on. Here is a list of some of the functions that the “client” in a client/server environment can perform:

- ◆ **Input validation.** In general, this is more than just validating that the field is of the correct type; it can include range checking and even table validation.
- ◆ **Data processing.** The client may be responsible for processing the returned data into a viewable form such as a pie chart or graph.
- ◆ **Menu processing.** Input values for a particular field may be chosen from a local table that is periodically downloaded from the server.
- ◆ **Application processing.** The client may perform some data conversion or application processing.

- ◆ **Compression.** Data compression or decompression may be performed by the client. This is especially important with video information.

When comparing client/server systems to GUI/server systems, consider the functionality of the client. By offloading work to increasingly more powerful clients, the performance of the entire system can be improved. As you see later in this chapter, it is also possible to offload the power of the server to a second-tier system.

Review of the Evolution of Computing

I hope the preceding discussion has conveyed to you my views about what defines a system and how a client/server system differs from a terminal or a GUI/server system. By understanding how the client operates and what its functions are, you can better see the ways that client performance is affected by such things as CPU, memory, and I/O capacity.

As the power of the client increases, so are the demands on it by new and more powerful applications. Those of us who have been in the computer industry for a while have seen the never-ending cycle of hardware and software: As new and faster hardware is introduced, new and more powerful software is developed to take advantage of the additional resources. There will never be a time when there is enough CPU power, enough memory, or enough I/O capacity to run our applications. This is what keeps the computer industry moving ahead; this is what keeps us going.

Two-Tiered and Three-Tiered Models

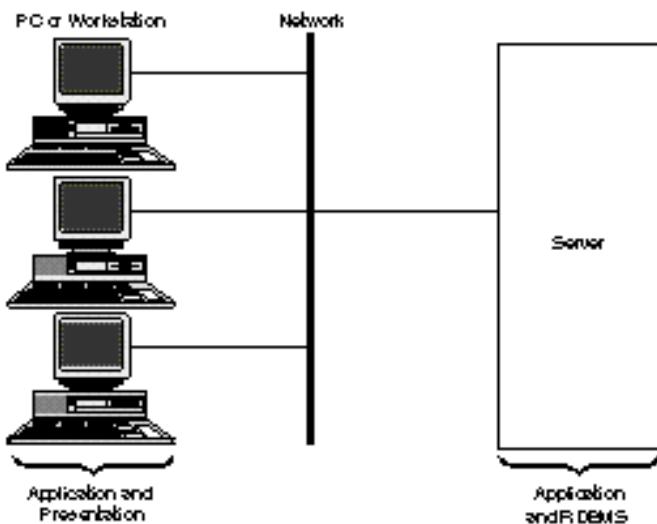
From the client/server computing model have evolved two different types of computing models: the two-tiered model and the three-tiered model. The two-tiered model is more like the traditional client/server model; the three-tiered model adds a level of processing known as the *application server*, which handles some of the functions of the client as well as some of the functions of the server.

Two-Tiered System

The two-tiered system is the traditional client/server system described earlier in this chapter. This system consists of one or more servers and a client. The client system runs part or all of the application; the server provides the RDBMS functionality. This traditional client/server system is shown in Figure 33.5. In this manner, the client and the server work together to provide the needed functionality to the user community.

Figure 33.5

The traditional two-tiered client/server configuration.



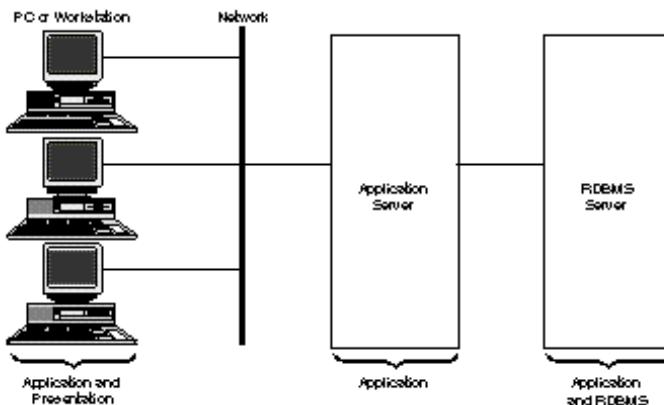
This model is typical for most client/server configurations today. However, I believe that many new client/server applications will soon move to the three-tiered model, described in the following section.

Three-Tiered System

The three-tiered client/server system uses the same client configuration as the two-tiered model but pushes some of the application logic away from the server and onto another machine. This third machine is sometimes known as an *application server* (see Figure 33.6). The application server is responsible for running much of the application code related to the server, freeing the server to handle RDBMS activities.

Figure 33.6

The three-tiered client/server configuration.



There are several benefits from moving the application away from the server:

- ◆ **Better performance.** By splitting the application and RDBMS processing onto separate machines, each can be tuned for their specialty.
- ◆ **Reduced contention.** In this model, because you do not need as many SQL*Net connections into the database server, contention may be reduced.
- ◆ **Resource reduction.** Because there are fewer SQL*Net connections into the database server, memory consumption on the server is reduced.
- ◆ **More flexibility.** Because the application is removed from the server, there is more configuration flexibility: You are not limited to one application server.
- ◆ **Increased control.** Because the application servers are handling requests from the users, queuing can be used to control the access to the database server.
- ◆ **Functional separation.** By separating the function of the database server from the application server, each can be maintained and serviced separately. Multiple application servers and database servers can provide for redundancy.

NOTE: These reasons lead me to believe that the three-tiered architecture will become more predominant in large installations with large applications. Some applications such as SAP financial systems are already taking advantage of the three-tiered architecture. Other applications such as Transaction Monitors have been in use for many years, making applications three tiered.

Client Bottlenecks

Now that the client has been defined, you can determine what affects the performance of the client system. The client system is responsible for running the client application and presenting the results to the user. The client application consists of the following parts:

- ◆ **Network connection.** This consists of the physical layer, the OS layer, and the SQL*Net layer. The SQL*Net layer may be shared with ODBC.
- ◆ **Application.** This is the part of the application that does the computation or processing of data.
- ◆ **Presentation services.** This part of the application is responsible for presenting the finished data to the user in the desired form.

Any of these areas can cause a bottleneck in the client system. The following sections look at these areas and determine what components can affect performance.

Network Performance

The performance of the network itself is usually not a problem for the client. The operating system vendors and Oracle have optimized their network components. There may be a problem, however, in exceeding the limits of the physical components. Judge this possibility on a case-to-case basis. Here are a few things you can look for to make this determination:

- ◆ Do you have out-of-date network components? If you are running on a Pentium client and using an 8-bit network card, consider upgrading.
- ◆ Is there an extreme amount of traffic on the network? By using a network analyzer, you can determine if you are reaching the limits of your network.
- ◆ Do other network activities seem sluggish? If the network seems slow with other activities, you may have a problem such as bandwidth that has been exceeded or too many collisions.

Typically, the network is not a problem; however, if any of these symptoms exist, you may want to take action by upgrading network components.

Application Performance

The application can be the cause of a bottleneck if it is not properly designed and tuned. Many older development tools were very inefficient and slow. By developing an optimized application, performance can be greatly enhanced. Here are some ways to determine whether your application is a bottleneck:

- ◆ Is the application compiled or interpreted? Compiled applications typically have less overhead and run faster.
- ◆ Does the client take more time to process than the server? By timestamping the application, you can determine how much time is spent in the application and how much time is spent waiting for the server.
- ◆ Does the application seem sluggish? Do simple operations such as data input seem too slow? If so, maybe the application is inefficient or the client hardware is insufficient for the task.
- ◆ Are the SQL statements efficient? Use SQL Trace and EXPLAIN PLAN to check the efficiency of these statements.
- ◆ Do other users have the same problem? Because the application is deployed to many clients, do they all see the problem or is it local to certain users? If only certain users have the problem, this may indicate that a specific client is at fault.

By making a determination about how efficient the application is, you may be able to determine whether it is the cause of the bottleneck.

Presentation Performance

Because many of these new client/server applications involve audio and video presentation, you should check whether you have inefficiencies in these presentation services. A graphics adapter that is too slow may be the cause of performance bottlenecks in video presentations. Many adapters can handle high resolutions and many colors but at a loss of performance. This may be a bottleneck for you.

Client Hardware Performance

Several components of the client hardware may be the cause of a performance bottleneck:

- ◆ **CPU speed.** An insufficiently powered CPU can be the cause of sluggish performance.
- ◆ **Insufficient memory.** A client that does not have enough memory will page or swap, which severely degrades system performance.
- ◆ **Slow network.** A slow network component can cause delays in transmitting and receiving data; however, this is not usually an issue for the client.
- ◆ **Slow I/O performance.** I/O is an issue only when loading the application—unless paging and swapping is occurring.

Any of these conditions can cause a bottleneck. However, the most common cause for performance problems on the client machine is the application.

Summary

This chapter began analyzing the client system and what could cause performance problems in it. It started out by defining what a client system is and reviewed the history of RDBMS applications and how they evolved into what we know as client/server computing. The chapter also introduced what I believe to be the next step in the evolution of client/server computing: the three-tiered client/server model.

Then the chapter looked at possible areas of client bottlenecks and a few ways to determine whether you are having problems in those particular areas. In debugging problems on the client, you frequently find that the client application is at fault because of inefficient SQL statements or problems in data processing.

Be sure that your client hardware has sufficient CPU and memory horsepower to handle sophisticated applications. With today's client machines, if you have enough memory, you should be able to achieve a good level of performance.

Chapter 34

Tuning the Client System

This chapter examines the client system and how the client operating system and the client hardware work together to serve the user. In the last few years, the power typical users have on their desks has increased dramatically. In the early 1980s, it was rare to see users with actual CPUs on their desks; more likely, they would have used terminals connected to a mainframe or minicomputer.

By the early 1990s, it was rare not to see a PC or workstation on most users' desks. Today, the power of these PCs and workstations has grown to where the raw CPU horsepower is equivalent to or greater than that of the minicomputer that may have serviced the entire office only 10 years ago.

As the power of these machines continues to grow, each user will soon have computer capabilities only dreamed of in the early 1980s. With this new power come new applications that take advantage of the power. This leapfrog effect keeps the computer and software industries continually improving their products.

In the early 1980s, the first PCs were just becoming popular, as was the MS-DOS operating system. At that time, there were few applications that ran on MS-DOS; those that did exist were amazing innovations. I remember that my cousin (who worked at IBM at the time, and still does) showed me a DOS program that calculated mortgage payments based on the input of a principal amount and an interest rate. At that time, having such a “power program” running at home was amazing. Remember: this was only 15 to 20 years ago.

As the power of the computer continued to grow, better and better software continued to be introduced to take advantage of that hardware. At the same time, the new software quickly made obsolete the old hardware. With the introduction and popularity of Microsoft Windows, Graphical User Interfaces (GUIs) became the standard.

Today, applications are being developed for the PC that a few years ago could not have been considered. With Microsoft Windows 95, it is now possible for the home user to take advantage of 32-bit applications that are more powerful than any ever available.

In fact, I frequently use Personal Oracle7 for Windows 95 at home for small database activities (such as my dogs database). Applications such as Personal Oracle7 and Oracle Power Objects are just a few examples of how software has been increasing in power to take advantage of newer and faster PCs.

Is there a point to all this or am I just rambling? The point I am trying to make is that the PC or workstation you have on your desk, if not already obsolete, soon will be. This is not a bad thing, just an indication that the applications available today to improve your productivity and your job are improving and will continue to improve. Because this is true, the capacity of the PC on your desk will improve also.

So what does this have to do with tuning the client system? As we continue to improve applications and client software, it is important to optimize the client to take advantage of all its capabilities. The client machine you are using today may seem very powerful but in a few years, when bigger and more powerful software is available to run on it, you will think that it is way too slow.

By optimizing the client system and finding ways to increase its capacity, you can lengthen the life cycle of these machines and provide better performance for your client applications. By optimizing the performance of the client machine, you can avoid costly upgrades until they are absolutely necessary.

This chapter looks at both the client operating system and the client hardware. It describes ways of improving client performance and how the client can be upgraded in the most effective way. Although upgrading a client machine may be the only way to improve a poor-performing client, remember that *upgrading* does not refer only to hardware but sometimes to software as well.

The chapter begins by looking at the Microsoft operating systems: Windows NT, Windows, and Windows 95. Later in the chapter, you look at ways to optimize UNIX systems as client systems. The chapter then describes some ways you can optimize and upgrade your systems hardware to provide better performance.

Windows NT

Microsoft Windows NT is very quickly gaining in popularity as a client operating system for the corporate user. Windows NT comes in two varieties: NT Workstation and NT Server. NT Server has more functionality—especially in terms of management tools—but for a client OS, Windows NT Workstation is sufficient.

Windows NT is a multiprocessor operating system; it can take advantage of scaleable performance increases by adding CPUs. Even though performance can benefit from the addition of processors, one CPU is usually sufficient for a client machine.

The NT operating system provides functionality such as 16-bit Windows application support and a Graphical User Interface (GUI). (The GUI is one of the prime reasons users have embraced Windows NT.) Other features that have made Windows NT a corporate standard include security features and robust management tools.

Tuning Memory

Perhaps the one disadvantage of the Windows NT operating system is its appetite for memory. Although you can run Windows NT Workstation in 16M or 24M of RAM, a minimum of 32M of RAM is recommended to run optimally. By using the NT performance monitor, you can check your memory usage and the amount of swapping that is occurring. Remember that swapping severely degrades performance.

One way to reduce your memory usage is to remove unnecessary services. By including only the network protocols you will be using, you reduce the overhead incurred by those additional protocols and reduce memory consumption.

Use the control panel to turn off any services you are not using. Doing so reduces memory usage and CPU overhead. By turning off all the services you will not be using, you can increase the performance of the system.

16-bit Applications

Although 16-bit applications will run on Windows NT, they lack the efficiency and robustness of 32-bit native applications. If possible, migrate your applications to 32-bit NT applications rather than using 16-bit Windows or DOS applications. Newer 32-bit applications are portable between Windows NT and Windows 95.

I/O Performance

I/O performance is not usually an issue for the client machine. The typical application is loaded into memory and does not use the I/O subsystem unless swapping is occurring. If your machine is swapping, performance is severely degraded and faster I/O will not significantly help.

Review of Windows NT

Microsoft Windows NT is a high-performance, robust workstation OS that is fast becoming a corporate standard. Windows NT does not require significant tuning to perform well. The best thing you can do to help improve client performance (besides tuning your application) is to reduce unnecessary memory usage to make more memory available for the application.

Microsoft Windows 3.1 and Windows for Workgroups 3.11

Microsoft Windows 3.1 and Windows for Workgroups 3.11 have been in production for quite some time. Microsoft Windows is currently the most popular graphical operating environment. Although Microsoft Windows is a very good operating system for home use, I recommend that you use Windows for Workgroups if you are running in a networked environment. Windows for Workgroups provides better networking than Microsoft Windows. Windows for Workgroups also provides better 32-bit support than Microsoft Windows 3.1.

Memory

As with Windows NT, the major concern with the Windows or Windows for Workgroups client is the lack of memory. Because Windows and Windows for Workgroups are both virtual-memory systems, it is possible to run much larger applications than will fit in memory. Although this is possible, it causes swapping, which degrades performance.

Unlike Windows NT and Windows 95, the Windows and Windows for Workgroups operating systems do not come with a performance monitoring tool that shows you when you have run out of memory. One way you can determine your memory usage under Windows 3.1 or Windows for Workgroups is to invoke a DOS shell and run the MEM command. The MEM command shows how much of your memory is currently in use.

Network

Another way to reduce excess memory usage is to reduce the number of network protocols currently loaded. Only load those protocols you plan to use. By reducing the number of excess protocols, you reduce the need for memory and CPU usage.

Review of Windows and Windows for Workgroups

When you use Microsoft Window 3.1 or Windows for Workgroups as your client operating system, the best opportunity for performance enhancement is to ensure that you have sufficient resources by reducing unnecessary overhead. By reducing this overhead, you eliminate unnecessary use of memory so that you can use that memory for your application. If at all possible, avoid swapping; swapping severely degrades performance.

Microsoft Windows 95

The recently introduced Microsoft Windows 95 greatly improves the Windows operating system. Windows 95 includes integrated 32-bit support, built-in networking, and other optimizations. If you currently run on Windows 3.1 or Windows for Workgroups, seriously think about upgrading to Windows 95 or Windows NT.

Windows 95 is robust and performs well but, like Windows NT, has an appetite for memory. Unless you have 12M or 16M of RAM, do not consider Windows 95. If you have the resources, there are many great features of Windows 95 that can help you optimize your system as an Oracle client:

- ◆ **Built-in networking.** The networking subsystem is built in as part of Windows 95 and has a very good Setup tool. The built-in networking can aid in the optimization of your network subsystem.
- ◆ **Integrated 32-bit support.** It is no longer necessary to add the Win32s add-on required under Windows. Integrated 32-bit support provides better 32-bit performance.
- ◆ **System analysis tools.** The Windows 95 control panel has an analysis tool that can determine whether your system is running optimally. (At least, it will tell you whether you are running any 16-bit drivers.)
- ◆ **System monitoring tools.** Unlike Windows 3.1 and Windows for Workgroups, Windows 95 comes with a system monitor, which display such things as CPU, memory, and I/O usage information.

The system monitor and integrated network subsystem features can substantially improve your optimization efforts. However, you must have enough system memory to run Windows 95.

32-Bit Support

Windows 95 is a 32-bit operating system that can run 16-bit applications. Even though it is possible to run 16-bit applications, doing so is not recommended. To take full advantage of Windows 95, you should run only 32-bit applications. Oracle provides 32-bit support for

Windows 95 that has been optimized for the operating environment. By taking advantage of the 32-bit support feature, you can improve performance.

Memory

With Windows 95 as with every OS, you must reduce unnecessary memory usage so that you can allocate as many resources as possible to the application. By reducing or removing any nonessential overhead from the system, you can increase the amount of memory available for the application. You must ensure that you have sufficient memory to run your applications effectively.

Network

By removing any unnecessary network clients, protocols, and services from the system, you reduce overhead. Running unnecessary network functions can increase CPU and memory overhead. With the Windows 95 network configuration utility, you can easily reduce these functions to the minimum required for your system.

Oracle Support

Oracle has recently provided native support for Windows 95. Use the Windows 95 version of SQL*Net to improve your application's performance. With native support for Windows 95, Oracle can take advantage of OS features and 32-bit support to optimize the product.

Review of Windows 95

I have been using Windows 95 for a while and am pleased with its performance and stability. Windows 95 can be a good-performing, robust client platform—if you have the resources. It is important that you have enough memory and CPU power to run an OS like Windows 95.

UNIX

The UNIX operating system is probably the most configurable of the clients described in this chapter. Even so, there is not very much you can configure to support client applications. As with the other OSes described in this chapter, with UNIX, it is important to make sure that you have the proper resources to run your applications.

The UNIX operating system requires more resources than Windows or Windows for Workgroups. The resources required to effectively run UNIX are somewhere between the resources required for Windows 95 and Windows NT. A PC UNIX system such as UnixWare or SCO UNIX needs a minimum of 16M of RAM and provides excellent performance with 32M of RAM.

Memory

As with the other operating systems described in this chapter, probably the most important factor with UNIX is the availability of memory. You should eliminate unnecessary processes and reduce other overhead. For the client OS, file-system buffers can be slightly reduced to allow for more user memory. Because reducing file-system buffers may slow down program loading, test this change carefully.

With the UNIX operating system, you can monitor memory usage by using `sar` or any of the graphical monitoring tools that may come with your OS.

Network

As you must do for the other systems described in this chapter, you should remove any unnecessary UNIX network protocols to reduce memory and CPU overhead. Monitor the network and increase stream resources if necessary. By reducing network overhead, you can increase the application's performance.

Review of UNIX

As with all the operating environments described in this chapter, it is important to monitor memory use with UNIX. If memory is a problem, try to reduce overhead and system memory consumption as much as possible. With any OS, you should make sure that your system has sufficient resources to run your application.

Hardware

Throughout this chapter, the recurring theme has been the availability of system memory. With client applications, the performance of the system is determined by three main factors. These factors, in order of importance, are as follows:

1. **The application.** If the application is inefficient or poorly written, there is nothing you can do to "tune" around this. A well-written application is the starting point in an optimized system.

2. **System memory.** The performance of a well-written and optimized application can be severely degraded by a lack of memory. If the application cannot reside in physical memory, the system will swap, which severely degrades performance.
3. **CPU power.** When you have optimized the application and made sure that there is enough memory, the application will run as fast as the CPU can process it. This is the ideal situation.

If you have sufficient resources and the application is optimized, your application should run as fast as the CPU can process it. When this is the case, the only way to run faster is to upgrade your processing power.

When upgrading computer resources, many people mistakenly upgrade their CPU. Memory is the prime cause of performance bottlenecks. If you are having client performance problems, check the system memory first. If you don't have sufficient memory resources, a faster CPU cannot improve performance.

Summary

This chapter looked at the client system from the perspectives of the OS and hardware. You have seen that system resources can be a problem if they are not sufficient for the required task; you also learned about some upgrade alternatives. When upgrading your system, consider upgrading your software and application as well as your hardware.

If you upgrade to a 32-bit operating system with additional features, you not only achieve a more robust system, you reap other benefits of a newer OS. Keep in mind, however, that you may have to increase the amount of memory in your system to support the new OS.

If you upgrade your client hardware, make sure that you upgrade the correct components. Upgrading a CPU in a system with a memory bottleneck will not improve performance. You may find that with a little more memory, system performance is greatly improved.

Chapter 35

Using GUI Builders

To rapidly deploy applications in a variety of different environments, it is often more efficient to use GUI application development tools rather than writing applications from scratch. This chapter describes some of the application development tools currently available.

This chapter describes some of the applications development tools currently available. There are several advantages to employing GUI development tools:

- ◆ **Rapid deployment of applications.** By using application development tools, you can develop functional applications very quickly.
- ◆ **Applications can be deployed in a variety of environments.** Some tools allow the same application to be compiled for Windows, Mac, and UNIX.
- ◆ **Applications can be easily modified.** Development tools can facilitate the transition of development from one developer to another. By knowing the tools, it is easier for a new developer to pick up where other developers have left off.

Several excellent third-party and Oracle tools are available for application development. This chapter takes a general look at how application development tools can be optimized and then takes a specific look at some of the application development tools in use today.

In particular, this chapter examines the Oracle application development tools (Oracle Developer/2000 and Oracle Power Objects) and these popular third-party tools: PowerBuilder from Powersoft, Delphi from Borland, and SQLWindows from Gupta.

Tuning the Application

When you develop an application using an application development tool, you can do quickly what would take years to do if you programmed by hand. The function of application development tools falls into two areas:

- ◆ **GUI development.** This involves building the Graphical User Interface and includes determining what data is to be displayed and how it is to be presented.
- ◆ **Database access.** The second function of these tools is to access the database and perform certain functions based on what the application is designed to do.

It is this second function that is most important to the focus of this book. This doesn't mean that the GUI cannot be a bottleneck—it certainly can. But there is little you can do to speed up the GUI portion of your application (typically, you cannot configure or modify that portion of the application). You can, however, modify the SQL statements generated by these GUI builders. So that is where we will focus our attention.

First-Generation Graphical Application Development Tools

When the first generation of application development tools was introduced, the tools were notorious for creating good-looking applications that were extremely slow. There were several reasons for this:

- ◆ **Badly formed SQL.** These early tools typically built the SQL statements dynamically at run time and usually were not very efficient.

- ◆ **No stored procedures.** The early tools usually did not take advantage of stored procedures, nor did they help you develop them.
- ◆ **Interpreted code.** These tools built the application using an interpreted language, which caused the application to execute slower than would a compiled version.
- ◆ **Reliance on ODBC.** Many of these tools required the use of ODBC rather than using a native Oracle interface; ODBC reduced performance.

Modern Graphical Application Development Tools

As graphical application development tools became more mature, many of the early performance problems were eliminated. In fact, now the features of almost all these application development tools include the following:

- ◆ **Improved SQL.** The quality of the SQL statements generated by these application development tools has increased dramatically since they were first introduced. And you now have the option of modifying the SQL statements manually.
- ◆ **Stored procedures.** Either through the development tool itself or by modifying the SQL statements manually, you can take advantage of stored procedures.
- ◆ **Compilation options.** Now, not only can you quickly run your application while developing it, you can compile it for deployment.
- ◆ **Native RDBMS support.** The development tools now allow you to use ODBC or a more efficient direct connection using SQL*Net.

Because of the improvements in the tools, there are no longer any specific ways of magically tuning the applications these tools create. This chapter shows you ways to access the SQL statements within the applications. Once you have access to those statements, you can determine whether these statements are efficient and can be improved.

How To Test and Improve Automatically Generated SQL Statements

Probably the best way to go about tuning SQL statements that have been generated by an application development tool is to extract the SQL statement from the application and tune it separately. You can do this using SQL*Plus and tools such as SQL Trace and EXPLAIN PLAN. Once you have optimized the SQL statement, you can insert it back into the application.

CAUTION: If you take this approach, make sure that you have not altered the return values in any way. When the SQL statements were created, the application was modified to expect certain data back from those statements. By changing the return values, the application may no longer function properly.

By extracting the SQL statements from the application, you can optimize them separately from the application itself. By timing the responses from Oracle, you may be able to determine how much time the application takes to process the GUI. Remember that these values are only approximations and may not be extremely accurate.

The following sections describe some of the application development tools on the market today and how to tune the applications they create for optimal performance.

Oracle Tools

Oracle offers several tools for the development of applications. Among them are the Developer/2000 tools, the Designer/2000 tools, and of course, Oracle Power Objects. These tools, like the third-party tools discussed later in this chapter, are all excellent products and should work very well for you.

Oracle has designed Developer/2000 and Oracle Power Objects to compete in different market segments. Although both products develop applications that run in the Microsoft Windows environment, Developer/2000 is more flexible and can create the same application in Windows, Macintosh, Motif, or character mode. Developer/2000 is also a much more complex and sophisticated tool; as such, it is more expensive and a little harder to operate.

Which tool is right for you really depends on the application you are writing and the environment you are writing it for. If your application must be deployed in several different environments, and more than one person is developing the application, you probably should think about Developer/2000. If you are developing a small departmental application for the Windows or Macintosh environment, you may do very well with Power Objects.

The following few sections describe each of the Oracle tools. The focus is on how to tune the applications these tools create. The sections contain hints on how to add performance optimizations to these application developer tools. I do not provide in-depth explanations about how to develop applications with these tools (that is beyond the scope of this book). I will, however, show pieces of a simple application developed to access the sample DOGS table as an example.

Developer/2000

Oracle Developer/2000 is a suite of tools designed to rapidly develop full-featured applications. The Developer/2000 tools consists of the following components:

- ◆ **Forms Designer.** Developer/2000 uses a drag-and-drop approach to designing forms applications. This approach simplifies the task of developing forms.
- ◆ **Reports Designer.** Developing reports is simplified through the use of graphical development tools that allow you to easily develop graphical reports.

- ◆ **Graphics Designer.** The graphics features allow you to easily add various chart types to your application.
- ◆ **Procedure Builder.** The Oracle Procedure Builder is an integrated environment for editing, developing, and debugging PL/SQL programs, stored procedures, and database triggers.

A very important feature of Developer/2000 is its ability to add online documentation to your application. I feel that documentation is a crucial part of any application.

Another major feature of Developer/2000 is its ability to develop a graphical application and then deploy that application in the Windows, Macintosh, and Motif environments without having to redevelop the application.

Oracle Developer/2000 is designed to work in a team environment; this means that many users can share logic and user-interface objects. The team approach to application development is one of the things that makes Developer/2000 suited for large applications.

The following sections look at Oracle Developer/2000 from a performance perspective to see how you can optimize applications developed with this tool.

Forms Designer

Forms Designer is the Developer/2000 tool for creating and deploying applications for the Windows, Macintosh, and Motif environments. The Forms Designer is key in quickly deploying applications in the Developer/2000 environment.

Applications developed with the Forms Designer are typically fairly optimized. To look at the SQL statements generated for a particular statement block, use the following steps:

1. Choose the block in which you are interested from the Object Navigator (see Figure 35.1).
2. Pull down and select the PL/SQL Editor from the Tools menu.

A PL/SQL window opens, which you can use to view or modify the SQL statements used in the Forms Designer (see Figure 35.2).

Once you select the PL/SQL Editor, you can view both triggers and program units on several levels. The Forms Designer takes advantage of triggers for doing much of its programming.

By using the Object Navigator and the PL/SQL Editor, you can modify all the SQL statements used in the application. In this way, any inefficiencies you may find—or optimizations you may want to add—can be inserted into the code.

TIP: The PL/SQL Editor is a great way to add hints to your SQL statements. The hints you add can inform the optimizer of such things as any additional parallelism you want to include.

Figure 35.1

Selecting the PS/SQL Editor.

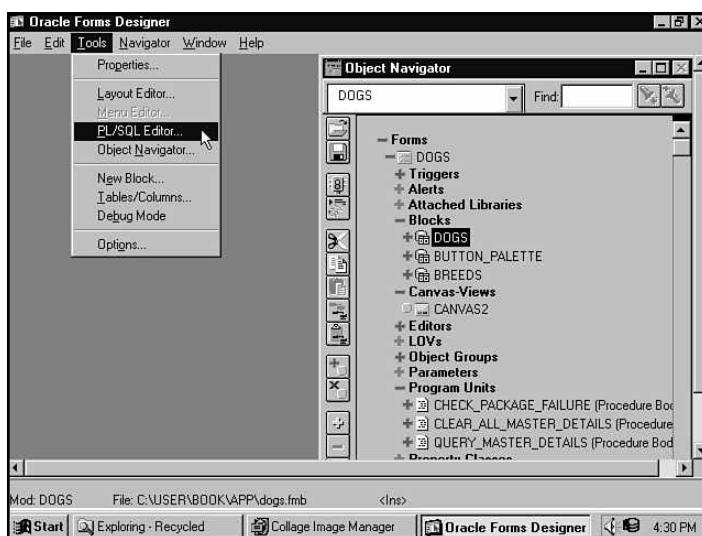
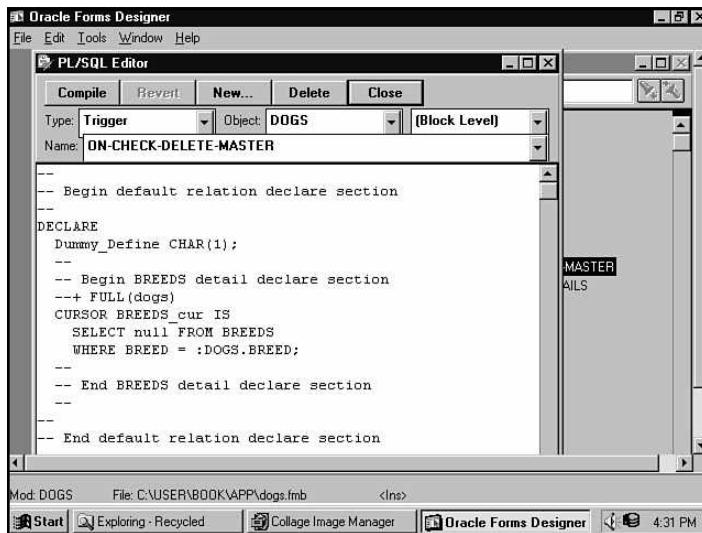


Figure 35.2

The PL/SQL Editor window.



It is entirely possible that there is no need to add additional optimization. However, by having the option to perform additional tuning and optimization on the SQL statements, you may be able to improve performance.

Reports Designer

Reports Designer is the Developer/2000 tool for creating reports applications for the Windows, Macintosh, and Motif environments. The reporting tool can build graphical and printed forms easily and quickly and is very configurable.

As with the Forms Designer, applications developed with the Oracle Reports Designer are typically fairly optimized. The actual query used in your report is defined as part of building the report. To modify this SQL statement, follow these steps:

1. Highlight the query you want to review by single-clicking it.
2. Pull down and select Properties from the Tools menu (see Figure 35.3).

The Query Editor opens; use this window to view or modify the query used in this report (see Figure 35.4).

Figure 35.3

Selecting the query to modify.

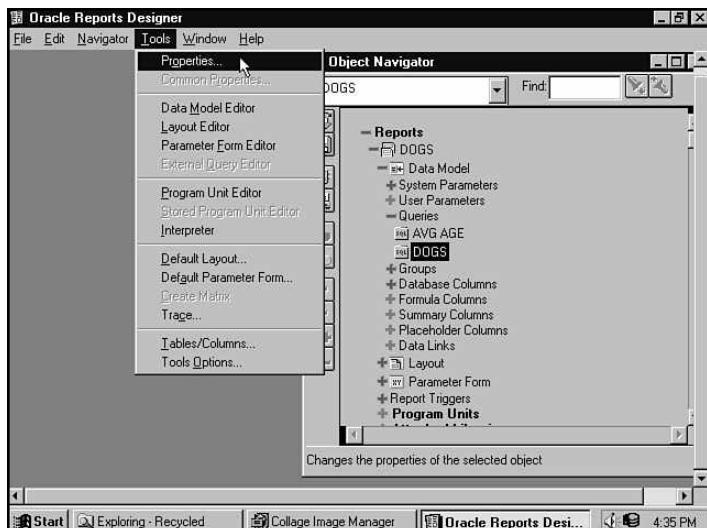
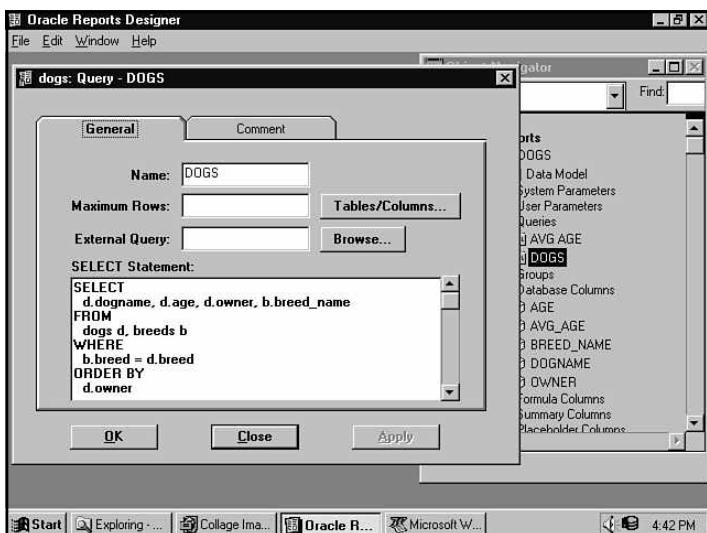


Figure 35.4

The Query Editor window.



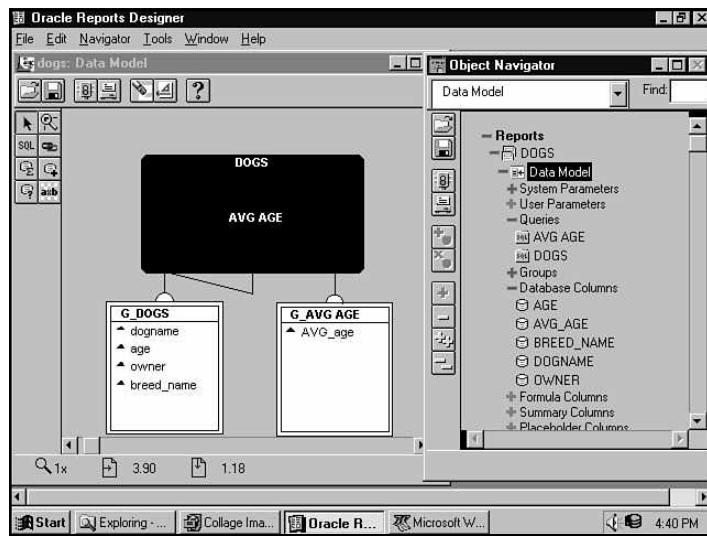
Once you invoke the Query Editor, you can view or modify the SQL statement for optimal performance.

By using the Query Editor, you can change your SQL statements to be more efficient. In this manner, you can completely optimize the application created by the Oracle Reports Designer tool.

TIP: As with the Forms Designer, the Reports Designer tool is a great way to add hints to your report queries. The hints you add can inform the optimizer of such things as any additional parallelism you want to include.

You can examine the inputs to the report using the Data Model Editor (see Figure 35.5) to get a fairly good idea of the number of queries going into the report. By optimizing all of them, you can make the report very efficient.

Figure 35.5
The Data Model Editor window.



It is entirely possible that there is no need to add additional optimization. However, by having the option to perform additional tuning and optimization on the SQL statements, you may be able to improve performance.

Graphics Designer

Graphics Designer is a tool for building graphical displays that can be used in a stand-alone fashion or in conjunction with the Oracle Forms Designer and Reports Designer. The Oracle Graphics Designer is similar to the Report Designer in that the creation of the query is part of the definition process.

Because you are creating the query that will be used in the creation of the final chart, you have complete flexibility to optimize this query. The steps necessary to modify a query once the application has been written are as follows:

1. Highlight the query you want to review by single-clicking it.
2. Pull down and select Properties from the Tools menu (see Figure 35.6).

The Query Properties Editor window opens; you can use this window to view or modify the query used in this report (see Figure 35.7).

Figure 35.6

Selecting the query properties.

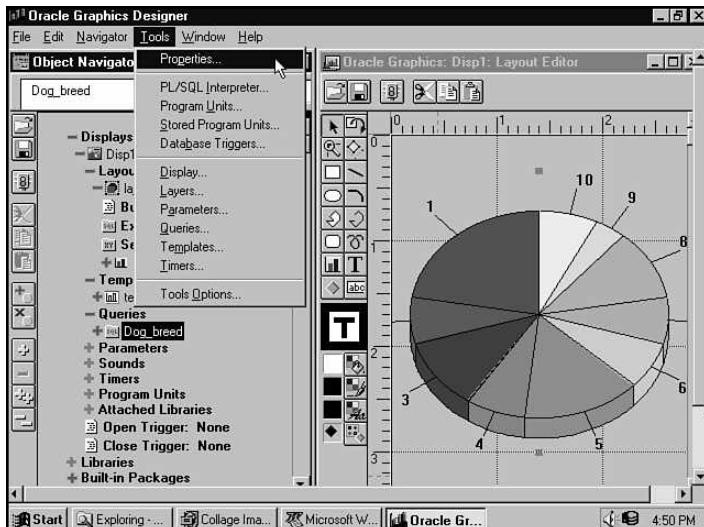
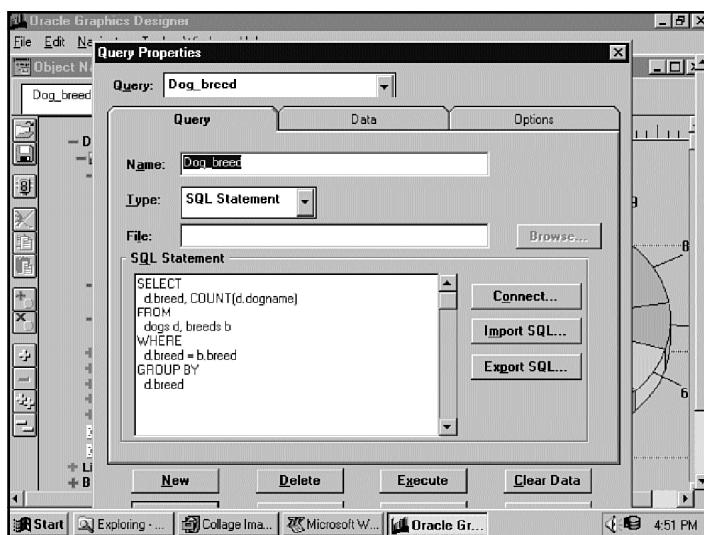


Figure 35.7

The Query Properties Editor window.



Once you invoke the Query Properties Editor, you can view or modify the SQL statement for optimal performance.

By using the Query Properties Editor, you can change your SQL statements to be more efficient. In this manner, you can completely optimize the SQL statements used in the application created by the Oracle Graphics Designer tool.

Procedure Builder

The Oracle Procedure Builder is an integrated environment for editing, developing, and debugging PL/SQL programs, stored procedures, and database triggers. The Oracle Procedure Builder is integrated into the Developer/2000 suite of tools.

The Procedure Builder does not create procedures or triggers for you; rather, it assists you in the development of those objects. As such, there is no type of optimization you can do to the Procedure Builder.

The following options are available from the Procedure Builder to assist you in the development of these objects:

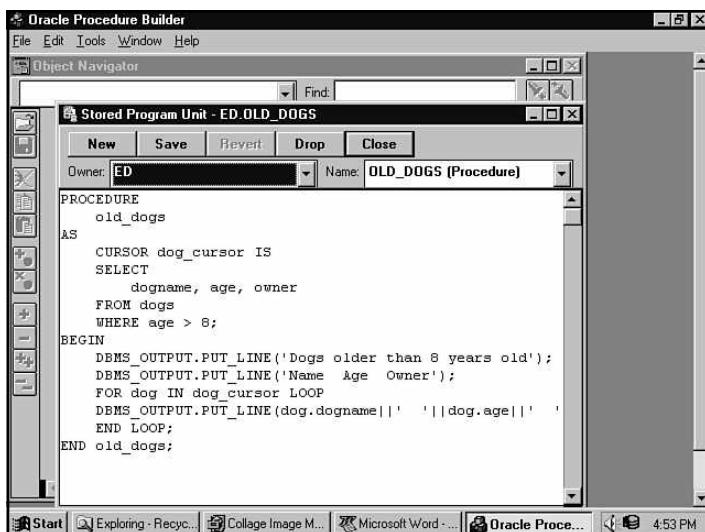
<i>Option</i>	<i>Description</i>
Program Unit Editor	Used for creating procedures, functions, and packages.
PL/SQL Interpreter	Useful for debugging specific statements.
Stored Program Unit Editor	Used to view and edit stored programs defined in your database (see Figure 35.8).
Database Trigger Editor	Used to develop and debug triggers (see Figure 35.9).
Source Code Control	The built-in Source Code Control lets multiple developers keep track of changing source code versions easily and effectively.

By using the PL/SQL interpreter, you have the ability to debug objects with the built-in debugging and tracing facilities. This can be useful for editing and obtaining information about stored programs in your database. The Database Trigger Editor allows you to view and modify the trigger body as well as change the trigger definitions.

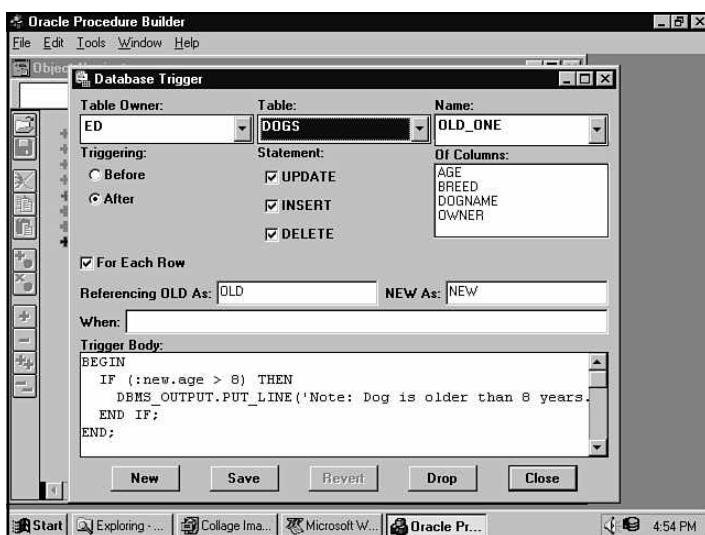
By using the various features included in the Procedure Builder (such as the development and debugging tools as well as the source-code control features), you can improve your productivity when developing applications.

Figure 35.8

The Stored Program Unit Editor.

**Figure 35.9**

The Database Trigger Editor.



Review of Developer/2000

Oracle Developer/2000 is a suite of tools designed to rapidly develop full-featured applications. As you have seen, these tools can be quite useful in the development and enhancement of forms, reports, and graphics objects. When you combine them with

Oracle SQL development tools, they can be extremely helpful when you are developing large applications.

If you are looking for something smaller and lighter than Developer/2000, you may want to consider Oracle Power Objects.

Power Objects

Oracle Power Objects is relatively new and gaining in popularity. Power Objects is an application development tool that can work with your Oracle database or with a small stand-alone database that comes with it called BLAZE. Power Objects is object oriented and can have you developing small applications in a matter of minutes.

Power Objects is very easy to use and powerful. By designing applications in an object-oriented fashion, you can easily reuse objects and save time. With Oracle Power Objects, you have the ability to build forms, reports, and classes. You can build on the classes, making Oracle Power Objects quite useful.

As with all the tools described in this chapter, Power Objects uses a drag-and-drop interface, which makes all the tools easy to get started with. The following sections describe the three major areas of Power Objects development.

Forms Designer

The Forms Designer is the part of Power Objects used to build applications. The Forms Designer uses a drag-and-drop approach to allow you to drag tables from the tables window into the form design window, as shown in Figure 35.10.

Figure 35.10

The Power Objects Forms Designer.



Applications developed with the Power Objects Forms Designer are typically fairly optimized. Unfortunately, Oracle Power Objects does not give you a view into the SQL statements that will be used to execute your transactions. However, you can view the SQL statements by using some user properties and the EXEC SQL command within some objects.

Reports Designer

The Reports Designer is the part of Power Objects used to build graphical reports. The Reports Designer also uses a drag-and-drop approach to allow you to drag tables from the tables window into the report design window, as shown in Figure 35.11.

Figure 35.11

The Power Objects Reports Designer.



As with the Forms Designer, applications developed with the Power Objects Reports Designer are typically fairly optimized. Unfortunately, as with the Forms Designer, Oracle Power Objects does not give you a view into the SQL statements that will be used to execute your transactions. However, you can view the SQL statements by using some user properties and the EXEC SQL command within some objects. This is not nearly as easy to do as it is with some of the other tools described in this chapter.

Classes

Classes are a key part of Oracle Power Objects. By using classes, you can reuse many of the objects you develop. If you use classes to contain reusable objects like labels, when you change the class, all other objects that use that class change automatically.

The Class design window also uses a drag-and-drop interface (see Figure 35.12). It is easy to create reusable classes very quickly. If you take the time to develop classes you can reuse, you can improve your efficiency in the long run.

Figure 35.12

The Power Objects Class Designer.



Power Objects gives you the ability to put these classes in libraries that can easily be used by others. Simply create a library and drop your classes or objects into it.

Review of Power Objects

As you have seen, Oracle Power Objects can be quite easy to use and can help you quickly develop and deploy applications. Unfortunately, it does not provide a view into the SQL statements that Oracle Power Objects is using in its transactions. You can use the `EXEC SQL` command in some objects by defining that as a function, but because you don't have the original SQL statement to work from, modifying them is difficult.

As a small-application builder, Oracle Power Objects is highly recommended. If you want to build a small application for home or work, this is the ideal tool. If you need to build a large, optimized application, I recommend that you use Developer/2000.

Third-Party Tools

Of the third-party tools available on the market today, several of the most popular are PowerBuilder from Powersoft, Gupta SQLWindows, and Borland Delphi. The following sections describe these tools with an emphasis on how to get the most performance out of them. All these tools are excellent products; I make no judgment about which tools are better or which will better suit your purposes. That choice is up to you.

Delphi from Borland

Borland's Delphi has grown in popularity over the last few years and is now one of the top-selling application development tools. Delphi is a graphical, drag-and-drop tool that is very easy to use and very powerful.

Delphi comes with ODBC drivers; by adding the SQL Links native drivers, you can connect to SQL*Net directly. It is much more efficient to build native Oracle applications than it is to use ODBC. ODBC essentially takes the calls to it and translates them to SQL*Net calls, causing unnecessary overhead.

Delphi consists of two main components: The forms developer is simply called Delphi, and the reports developer is called ReportSmith.

Delphi Forms Developer

As is true with the other tools described in this chapter, the SQL statements generated by Delphi are generally good. If you do need to alter the SQL statements to make them more efficient, you can do so easily. While in the forms editor, you can easily modify the SQL statement, as shown in Figure 35.13. To edit the SQL statement, follow these steps:

1. Highlight the TQuery box in the forms editor or select it from the pull-down menu in the Object Inspector.
2. Double-click the three dots next to the SQL selection.

This opens the String List Editor, where you can view or modify your SQL statements (see Figure 35.14).

Figure 35.13

Invoking the String List Editor.

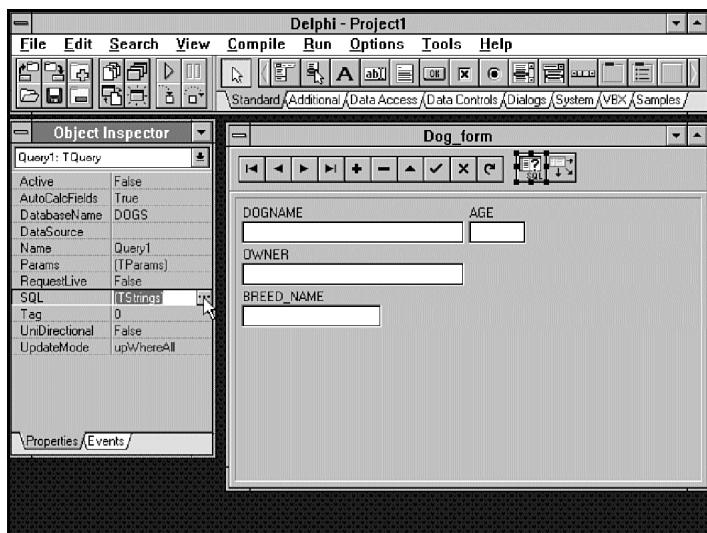
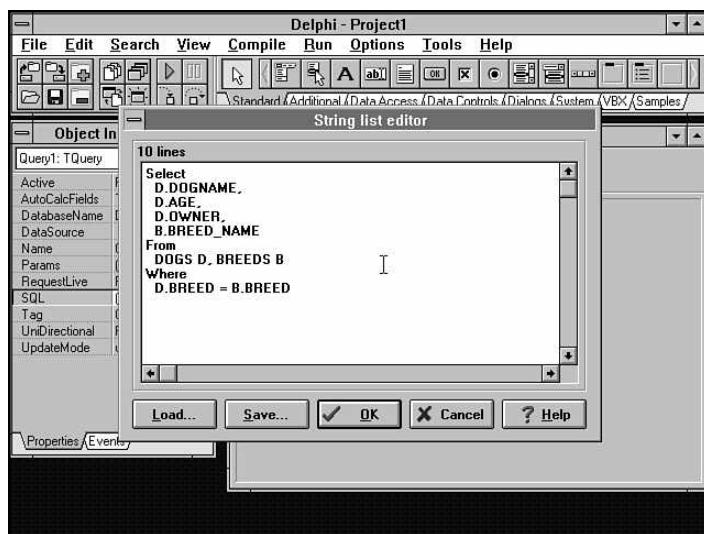


Figure 35.14
The String List Editor.



Once you open the String List Editor, you can modify and optimize the SQL statements that will be used in this application.

Depending on the complexity of your application and the database, you may not have to change the automatically generated SQL statements. But the capability is there if, for example, you want to change the SQL statement to invoke a stored procedure or function.

ReportSmith

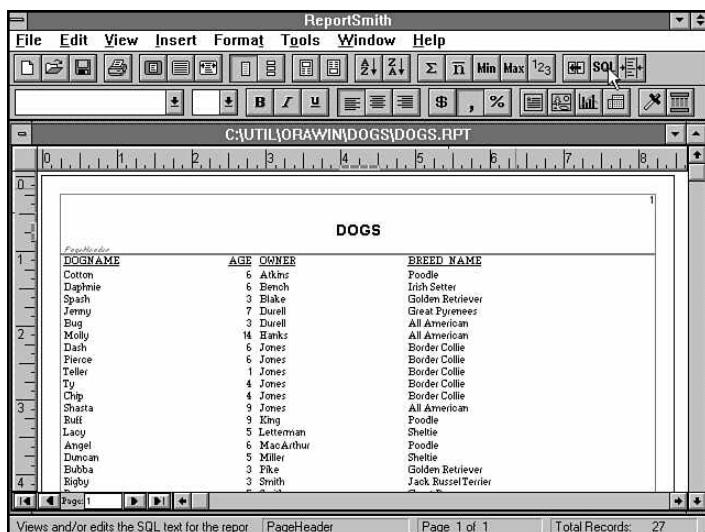
As with the Delphi forms development tool, the SQL statements generated by ReportSmith are generally good. If you have to alter the SQL statements to become more efficient, you can do so easily. While in ReportSmith, you can easily modify the SQL statement as shown in Figure 35.15. With ReportSmith, you can alter the SQL statements while you design the report or afterwards. To edit the SQL statement, follow these steps.

1. If you are altering the SQL statement for an already designed report, simply double-click the SQL icon in the toolbar at the top of the window (see Figure 35.15).
2. This opens the Report Query window, where you can modify the SQL statement as shown in Figure 35.16.

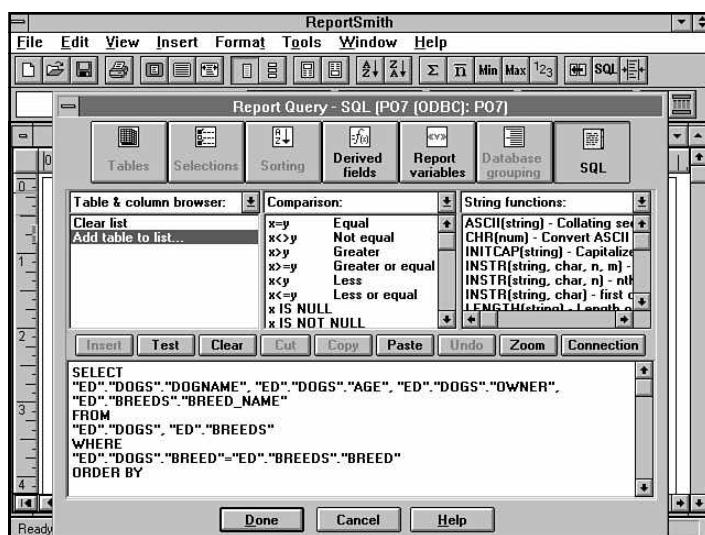
Depending on the complexity of your report, you may want to modify the SQL statements or add hints that can be used by the Oracle optimizer. Depending on the complexity of your application and the database, it may not be necessary to change the automatically generated SQL statements. If necessary, however, you can change the SQL statement to invoke a stored procedure or function.

Figure 35.15

Invoking the Report Query tool.

**Figure 35.16**

The Report Query tool window.



TIP: Depending on the size of the query and the amount of data to be scanned, you may see a huge performance increase by using the Parallel Query option. Use the Report Query tool to add parallel query hints to your SQL statements.

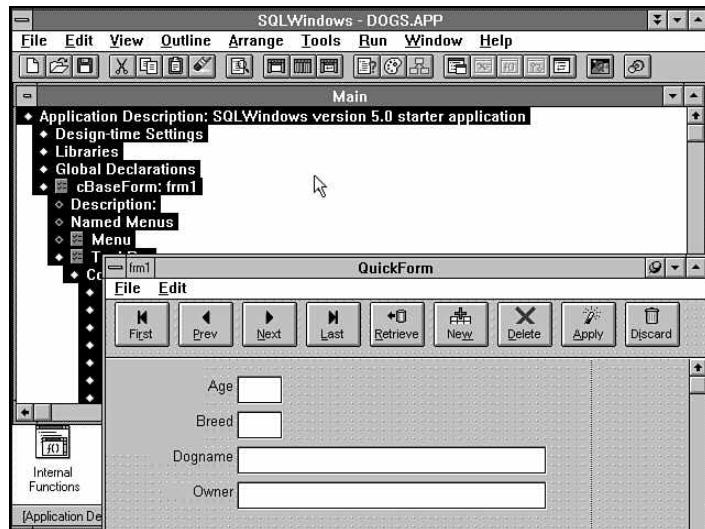
SQLWindows from Gupta

Gupta's SQLWindows is another very popular application development tool. As with the other tools, SQLWindows guides you through the development of an application using its QuickForms wizard. Using this tool, you can easily and quickly develop a usable application.

SQLWindows comes with ODBC drivers and recommends using them as the primary method of connection. The SQLWindows tool comes with a forms generator you can use to quickly deploy applications (see Figure 35.17). The forms tool is very easy to use and can have you building and running applications quickly.

Figure 35.17

The SQLWindows forms tool.

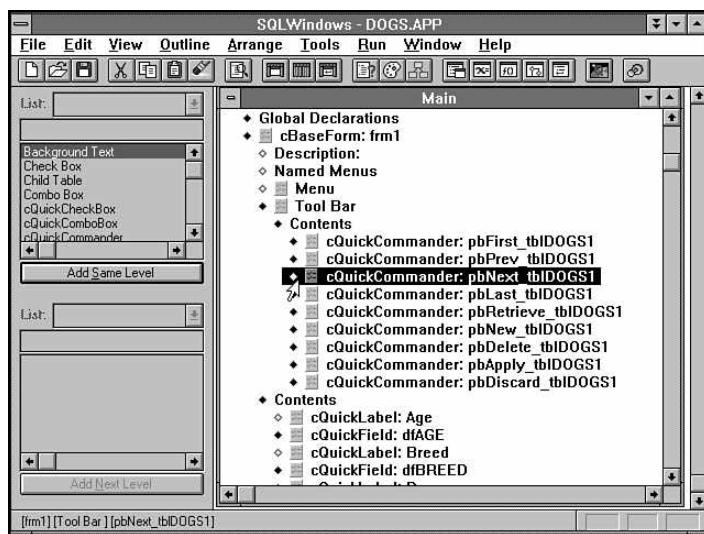


Although SQLWindows uses a scripting language, it is possible to use user-defined functions. To define your own functions or SQL statements, choose the object you want to modify from the main window. SQLWindows uses an outline presentation for developing the applications. By double-clicking the object, as shown in Figure 35.18, you eventually display the Message Actions.

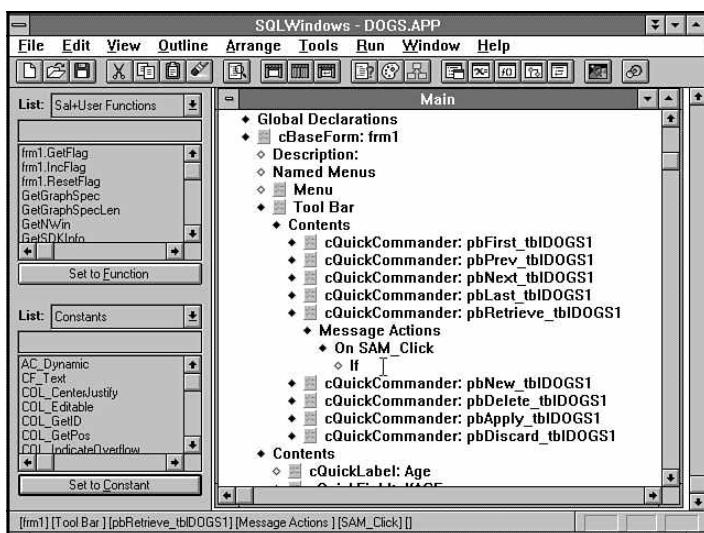
Adding a function to the Message Actions line allows you to override the default action and apply your own action (see Figure 35.19). At this point, you can define your own function or use a predefined SQLWindows function to optimize your SQL statements. As is true with Oracle Power Objects, it is difficult to modify the SQL statements in SQLWindows, and you don't have access to the SQL statements that SQLWindows uses by default.

Figure 35.18

The SQLWindows main development screen.

**Figure 35.19**

Adding functions to the Message Actions line.

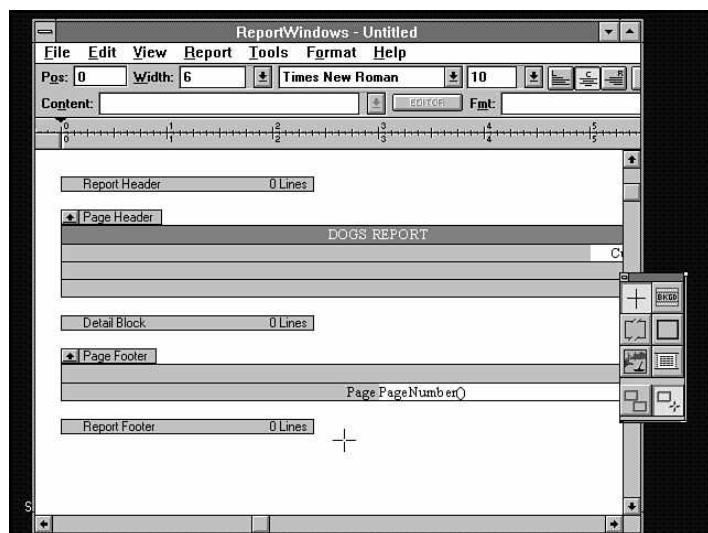


As with other tools described in this chapter, you can run in interpreted mode for debugging and development, but you can compile the application for deployment. In this manner, you can quickly develop and prototype applications, but your deployed applications are more efficient and faster because they are compiled.

For developing reports, SQLWindows comes with a report design tool called ReportWindows (see Figure 35.20). ReportWindows allows you to build custom reports very quickly.

Figure 35.20

The SQLWindows ReportWindows window.



As you have seen, SQLWindows has a slightly different way of looking at application development: it uses an outline format to develop the application. Even so, both of the tools in SQLWindows are very good.

PowerBuilder from Powersoft

PowerBuilder from Powersoft is by far the most popular application development tool. PowerBuilder was among the first to introduce these types of products.

PowerBuilder comes with ODBC drivers and native drivers that allow you to connect to SQL*Net directly. It is much more efficient to build native Oracle applications than it is to use ODBC because ODBC essentially takes the calls to it and translates them to SQL*Net calls, causing unnecessary overhead.

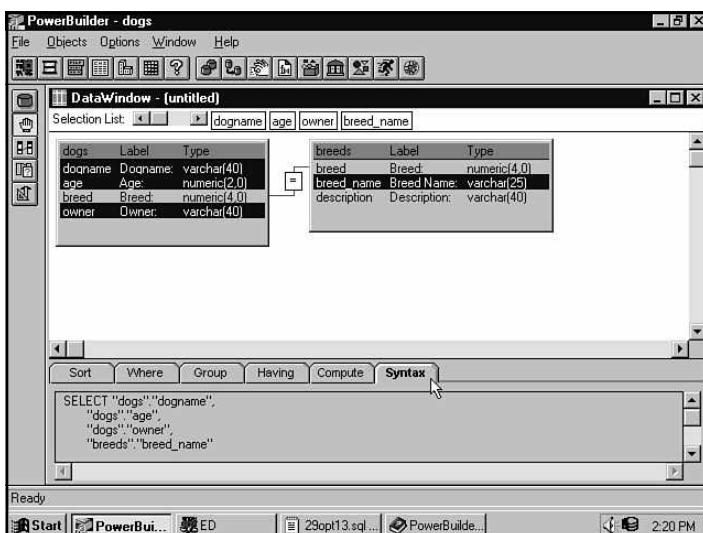
To build PowerBuilder forms, you use the DataWindow to describe the columns you want to use in the form. During this development process, the SQL statements you will be using to query this table are automatically available to you by clicking the Syntax tab (see Figure 35.21).

In this manner, you have instantaneous and simple control over the structure of your SQL statements. If you are working with a PowerBuilder application, you get to this window by clicking the DataWindow tool icon on the main toolbar (see Figure 35.22).

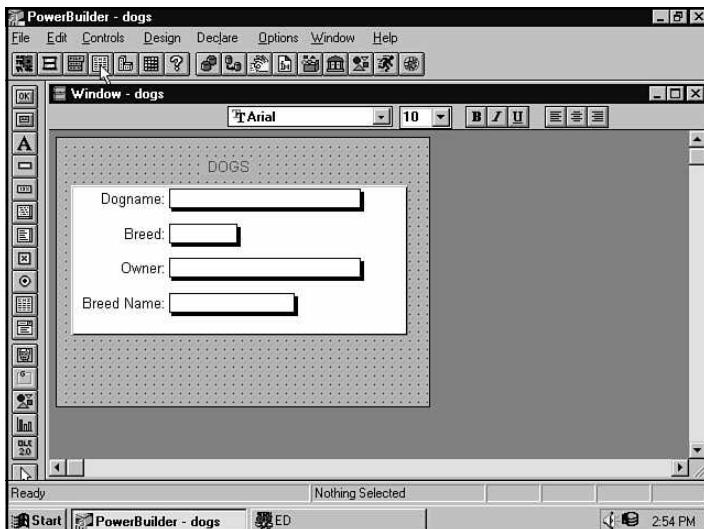
Once in the DataWindow, you can optimize and change the SQL statements to suit your needs. As mentioned earlier in the chapter, application development tools have evolved over the years so that now their SQL statements and the GUI are efficient and perform well. In most cases, there is no need to alter these SQL statements or the application to improve performance.

Figure 35.21

The PowerBuilder DataWindow.

**Figure 35.22**

Selecting the PowerBuilder DataWindow.



Summary

To rapidly deploy applications in a variety of different environments, it is often more efficient to use GUI application development tools instead of writing applications from scratch.

As you have seen in this chapter, there are several excellent third-party and Oracle tools available for application development. The chapter described some of these tools and explained how to modify the SQL statements generated by the tools to take advantage of optimization techniques.

You should now have an idea about how some application development tools work and how you can optimize the SQL code the tools generate. When you add the information in this chapter to the other chapters in Part V, “Tuning the Client,” you are well on your way to developing an optimized application and client system.

The next chapter looks at some ways to improve the overall performance of your configuration by taking advantage of middleware. By distributing the load among several machines, you may get better overall performance than with one large server.

Chapter 36

Using Middleware Products

This chapter examines the use of middleware to help improve the performance of your client/server configuration. The chapter begins by defining middleware and how you can use it in your applications. After this introduction, the chapter reexamines the difference between the two-tiered and three-tiered models, focusing on how applications can take advantage of these architectures. You also learn how some systems take advantage of application servers to improve overall system performance. Finally, the chapter discusses some applications you can use to take advantage of the three-tiered architecture.

What Is Middleware?

Middleware is a generic term used to describe any software or system used to facilitate processing as an intermediary between a client and server system. The term itself is not very descriptive except for the fact that these types of systems operate “in the middle” of the application.

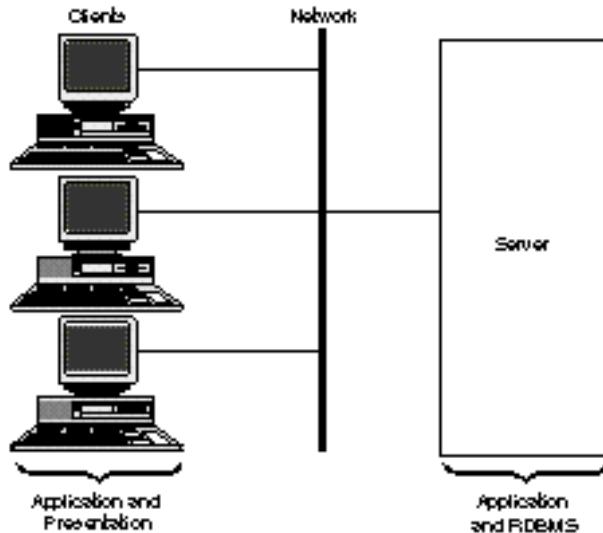
Middleware can take many forms, some of which involve off-the-shelf components and others that involve custom applications. Middleware is the key component in the three-tiered model introduced in Chapter 34, “Tuning the Client System.” The following sections describe the difference between the two-tiered and three-tiered models used in client/server computing and explain how the middleware components function.

Two-Tiered System Architecture

The two-tiered system is the traditional client/server system you are probably most used to. This system consists of one or more servers and a client. The client runs all or part of the application and the server provides the RDBMS functionality (see Figure 36.1). In this manner, the client and the server work together to provide the required functionality to the user community.

Figure 36.1

The traditional two-tiered client/server configuration.



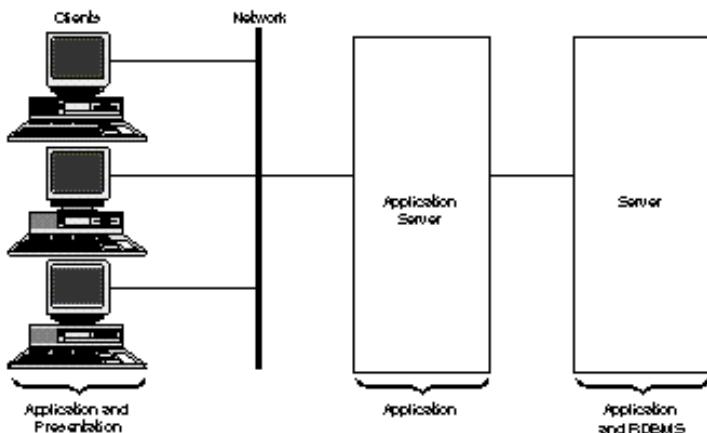
This model is typical for most client/server configurations today. However, I believe that many new client/server applications will move to the three-tiered model in the future because of its increased functionality and performance.

Three-Tiered System Architecture

The three-tiered client/server system uses the same client configuration as the two-tiered model but pushes some of the application logic away from the server onto another machine. The third machine is sometimes known as an *application server* (see Figure 36.2). The application server is responsible for running much of the centralized application code related to the server, freeing the server to handle RDBMS activities.

Figure 36.2

The three-tiered client/server configuration.



You can claim several benefits from moving the application away from the server:

- ◆ **Increased performance.** By splitting the application and RDBMS processing onto separate machines, you can tune each for their specialty. By tuning the RDBMS server for maximum throughput and the application for increased application response time, you can optimize each. You don't have to tune your RDBMS server to handle 1,000+ connections; let the application server do that.
- ◆ **Reduced contention.** Because you do not need as many SQL*Net connections into the database server, contention may be reduced. Each SQL*Net connection into the server takes certain system resources, including memory and other CPU resources needed to service requests. By reducing the number of connections, the overall load may not change but some of the overhead is reduced. Memory usage is reduced by reducing the number of connections; this saved memory can be used for more productive purposes such as additional database block buffers.
- ◆ **Increased configuration flexibility.** Because the application is removed from the server, you have more configuration flexibility. You are not limited to one application server; you can choose to have several application servers. For example, the RDBMS may service the entire company, but you may choose to split up the application servers for accounts payable, finance, accounts receivable, and so on.

- ◆ **Increased control.** Because the application servers handle requests from the users, you can use queuing to control the access to the database server. When you take advantage of TM features such as queuing, you can queue some tasks or serialize them to reduce contention.
- ◆ **Functional separation.** By separating the functions of the database server and the application server, each can be maintained and serviced separately. Multiple application servers and database servers can provide for redundancy.

These reasons lead me to believe that the three-tiered architecture will become more predominant in large installations with large applications. Some applications such as SAP financial systems already take advantage of the three-tiered architecture. Other application layers used as a middle tier (such as Transaction Monitors) have been in use for many years.

Application Servers

Application servers are in use in many installations throughout the world today. The term *application server* really describes any kind of system used to run an application that services clients and uses an RDBMS server. These types of systems can be used to take input from clients, process that input, and use the resulting data as input to the RDBMS server; alternatively, they can take input from the RDBMS server, process that data, and send it to the client.

The primary task of the application server is to offload server application tasks from the RDBMS server. Many of these tasks must be run on one system because of issues of data consistency and locking. Not all application tasks can be moved to the client; many tasks have to coordinate actions for many clients to provide for data integrity. The example in the following sidebar may help make things a little clearer.

Application Server Example

The prime example of an application server is a financial system. By offloading such services as accounts payable, accounts receivable, and finance to an application server, the tasks these services perform can be isolated from the RDBMS server. Of course, the RDBMS server must keep track of the data, but the act of processing the data can be offloaded. It would be very difficult to offload much of this work to individual workstations because of the coordination that must be performed within the finance department.

Within the different departmental machines, each of the services necessary to that department is processed on its individual application server. Those tasks that can be pushed out to the client (such as input validation, table lookups, and presentation processing) should be moved there. By segmenting the work into several different machines, each can be sized appropriately for its individual task.

How To Tune the Application Server

Tuning the application server depends on the function of the server. Typically, this type of machine is memory dependent. You must make sure that you have sufficient memory to avoid paging and swapping. The particular areas to monitor and tune on the application server include the following:

- ◆ **Memory.** Be sure that you have plenty of RAM to avoid paging and swapping. Because of the large number of users that might be connected, be sure to monitor at peak times. An insufficient amount of memory can drastically reduce performance.
- ◆ **CPU power.** Monitor the system CPU usage. If you are constantly running at 100 percent capacity and performance seems too slow, you may want to upgrade your processors; consider adding more processors if you have an SMP or MPP system.
- ◆ **Network.** If you constantly exceed 60 percent of your network bandwidth, you should add faster hardware or segment the network.
- ◆ **Application.** Your application may be tunable. If you use queuing mechanisms, you may have to add additional queues for certain functions. Take this advice on a case-by-case basis.

By taking advantage of application servers, you may be able to distribute the application load among several machines, increasing overall throughput. These architectures are new but are becoming increasingly more popular with larger applications. I believe that these kinds of distributed systems will continue to grow in the near future.

Transaction Monitor (TM)

Transaction Monitors (TMs) have been around for quite some time. Probably the most popular of the TMs is Tuxedo, which has been used for a number of years in many different configurations and applications. With the increasing popularity of client/server applications, TMs will become a necessary part of these applications.

To understand how TMs are used and how to tune them, you must first understand what a TM is.

What Is a TM?

A *Transaction Monitor* is a software product designed to facilitate communication and data processing between a client and server system. This system takes data input from the application, processes that data using a queuing mechanism, and submits that data to the RDBMS server. Here are some of the features of a TM you can use to improve client/server processing:

- ◆ **Name services.** By using the name services feature of the TM, clients can seamlessly move from one RDBMS server to another without knowing it. Users can request data and not even realize where the data is coming from.
- ◆ **Connectivity.** Users need not take the time to connect to multiple RDBMS servers. They need to connect only to the TM.
- ◆ **Queuing.** Transactions can be queued within the TM to reduce the number of concurrent transactions of any one type. The administrator has control over this queuing.
- ◆ **Multiplexing.** Many client connections can be multiplexed into a few connections that are made with the RDBMS server.
- ◆ **Priority management.** By queuing and multiplexing transactions, transactions can have their priorities controlled. A particular transaction can have priority over other types of transactions.
- ◆ **Distributed processing.** TMs provide distributed processing support such as a two-phase commit; they can be programmed to reroute requests to backup servers if a server fails.

To use a TM, you must write your applications to take advantage of the TM. Applications must be written in two parts: a client side and a server side.

The client-side code must perform these functions:

1. Take input from the client.
2. Determine what the transaction function is.
3. Queue the transaction based on that function.
4. Wait for the transaction to be processed.
5. Return the data to the user.

The server-side code must perform the following functions:

1. Receive the transaction from the queue.
2. Process the job based on the function of this module.
3. Submit the transaction to Oracle using SQL*Net calls.
4. Wait for a response from the RDBMS.
5. Return the completed job to the TM for the client-side process.

For example, you may have an application that performs account updates, status functions, and creation functions. The client-side process puts the transaction on one of three queues based on the function of that transaction. There can be any number of update, status, and creation processes to handle jobs on the queues. These processes are responsible for their particular function and process the data based on that function. In this manner, creation can be given a lower or higher priority than the other functions, and the number of account-creation tasks can be controlled.

By using a TM, you have an incredible amount of control over how your transactions are processed and scheduled. An easy-to-use interface makes programming the TM very straightforward.

When To Use a Transaction Monitor

Transaction Monitors have a variety of uses, including queuing, multiplexing, and distributed processing. For installations in which the number of client machines can be in the thousands with hundreds of servers, the number of network connections can be greatly reduced by using a TM. If you use Tuxedo in environments such as Windows NT and NetWare, you do not have to learn additional operating systems.

Tuning the TM and System

The process of tuning the system can be broken into two parts. You have a lot of control over how you tune the TM; you also must consider the OS.

Tuning the TM

The TM can be configured to provide a lot of control over how the transactions are processed. Some of the things you can configure in the TM include the following:

- ◆ **Priority.** The queue priority can be set to define a priority for the service. This priority is used relative to the priority of other services in the TM.
- ◆ **Number of queues.** You can set the number of queues you want to use. This capability is useful in limiting certain types of operations to reduce contention.

By configuring the TM, you can control and vary the amount of contention in the RDBMS. You can also give some transaction types more opportunities to run than others.

Tuning the OS for the TM

The key to tuning the OS for the TM is to ensure that there are sufficient system resources. TMs typically demand a moderate amount of shared memory and semaphores. Because there are usually many user processes connecting in from clients, you must have sufficient network resources and memory to handle these users.

Review of Transaction Monitors

The Transaction Monitor has been around for quite some time in many environments. With the increasing popularity of client/server applications and increasing numbers of distributed applications, the need for the TM is increasing. By offloading much of your processing and connectivity to these middleware machines, both the client and server can see reduced overhead.

Summary

This chapter examined the use of middleware and how it can help improve the performance of your client/server configuration. It started out by defining what middleware is and how it can be used in your applications. The chapter then reexamined the difference between the two-tiered and three-tiered models, focusing on how applications can take advantage of these architectures and how they can improve performance. Finally, you learned how a Transaction Monitor functions and how you can use a TM to reduce overhead on both the client and server machines.

By taking advantage of such things as application servers and Transaction Monitors, you can offload much of the work being done on one or two machines. By distributing this load, you enhance the performance of the entire application. By relying on one machine to handle everything, you reduce its efficiency because of the increased overhead required to do both application and RDBMS processing. When designing systems for performance, be innovative and look for new ways of solving problems. By taking advantage of innovative technology, you may be able to achieve results you have not yet even imagined.

Part VI

Tuning the Network

Chapter 37 What Affects Network Performance?

38 Tuning the Network Components

In Part V of this book, you saw how the application can be designed and modified to take advantage of SQL. You also looked at how to best tune the client operating system to take maximum advantage of the resources available to it.

Part VI, “Tuning the Network,” covers everything that is left over after you have tuned the server and the application. The two chapters in this part explain how you know whether you have network problems and what to do to solve the problems.

Chapter 37

What Affects Network Performance?

This chapter looks at the network itself: how the physical networking hardware operates and communicates with the device drivers and OS. Once you understand how the network operates, the chapter describes some of the things that affect performance and how to know when your network requires an upgrade to accommodate more capacity.

With the increasing move to client/server applications, the network itself has become more important. For your application to function properly, you may have to move large amounts of data between the client and server machines. If there is any delay in the movement of this data, caused by network contention or inefficiency, your overall performance suffers.

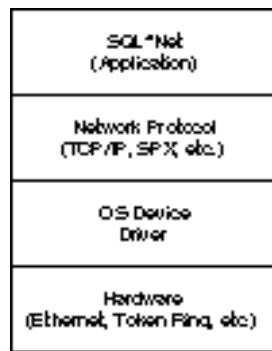
By understanding how the network hardware and software operate, you can understand what affects the performance of the network and why. The chapter starts by looking at the different types of network hardware and software available.

Network Architecture

The network itself is made up of many components, both hardware and software, and may vary in function. Several different types of networks are available on the market today, including Ethernet, Token Ring, fiber optics, and some new technologies such as ATM. On top of these different hardware layers lies the OS device driver, the network protocol layer, and finally SQL*Net itself (see Figure 37.1).

Figure 37.1

Network layers.



For the SQL request to be sent from the client to the server, the request must go through each of these layers—on both the client and the server. The following sections describe these components in more detail.

Hardware Components

At the lowest layer of the network subsystem lies the network hardware itself. This hardware can be of a variety of different types that provide different functionalities and characteristics. Probably the most popular of these network types are Ethernet, Token Ring, and fiber optics networks (FDDI).

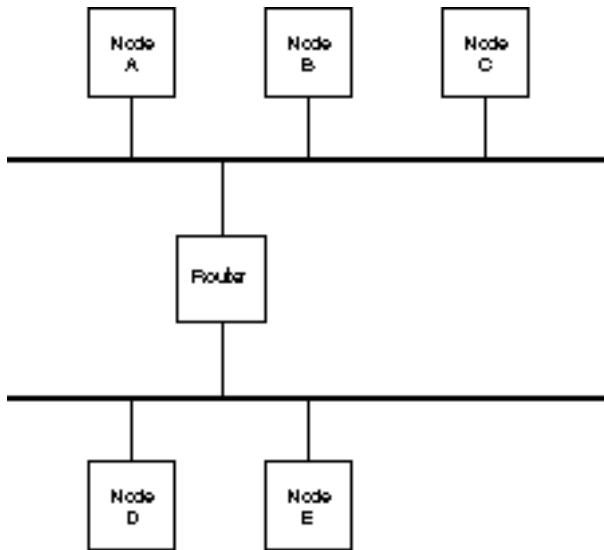
Ethernet

Ethernet is probably the most popular of the network protocols. Ethernet has been around since the early days of computer networking. The Ethernet standard includes various types of wiring (such as twisted pair and coaxial) and speeds (a standard speed of 10 megabits/second for most Ethernet and 100 megabits/second for the newer 100Base-T and 100Base-VG).

Ethernet is standards-based, which means that multiple vendors can build Ethernet NICs, hubs, routers, and so on that all work together, regardless of the brand. An Ethernet network is built around the idea that all machines on an Ethernet segment have equal access to the network. As long as the network is available, the NIC can use it (see Figure 37.2).

Figure 37.2

An Ethernet network.



The NIC sends the data in a structure called an *Ethernet packet* or *frame*. These packets contain not only the data network protocol information but also information the NIC has sent concerning the packet and the receiving NICs address. When a packet is sent, all NICs or routers on the network receive the packet. Each NIC performs a quick check; if the packet is not addressed for that NIC, the packet is quickly discarded. If the packet is intended for the NIC, further processing is done.

The Ethernet packet contains a *network packet* generated by the network protocol driver. Additional Ethernet information is placed around this network packet to create an *Ethernet packet*. The structure of the packet is determined by the network protocol being used.

When the device driver sends a network packet to the NIC to be sent across the network, the NIC checks to see whether there is any traffic currently on the network. If the network is busy, the NIC waits a while and then tries again. The packet is stalled waiting for the network and is referred to as a *deferred packet*.

When the NIC believes that the network is available, it sends the packet out. The NIC checks during the sending of the packet to make sure that no other NIC initiates a transmission at the same time. If another NIC had sent a packet out at the same time, a *collision* occurs. When a collision occurs, both NICs must wait a small amount of time and try to resend their packets. The contents of any sent packets that have experienced a collision are discarded.

Whenever a collision occurs and the packet has to be re-sent, performance is degraded a little. Although the effect of a single collision is very small and hardly noticed, the effect of thousands of collisions is noticeable. Collisions prevent the true bandwidth potential of an Ethernet network from being reached.

As the network is driven closer to its maximum bandwidth, the number of collisions increase exponentially. The more traffic there is on the network, the more likely it is that your packet will collide with another packet. Therefore, even though your network may be able to theoretically handle 10 megabits/second, it is unlikely that you will ever achieve this rate. These things make collisions more likely:

- ◆ **High numbers of connections.** If there are many NICs on the LAN segment, you are more likely to see collisions because it is more likely that a NIC will be transmitting.
- ◆ **Small packet sizes.** If the packet size is small, there will more likely be more packets, and the collision rate will increase.

With smaller packets, there is likely to be more wasted time between packets (that is, time during which there is no network activity). With larger packets, there is less inactivity on the network. Larger packets are more efficient and perform better.

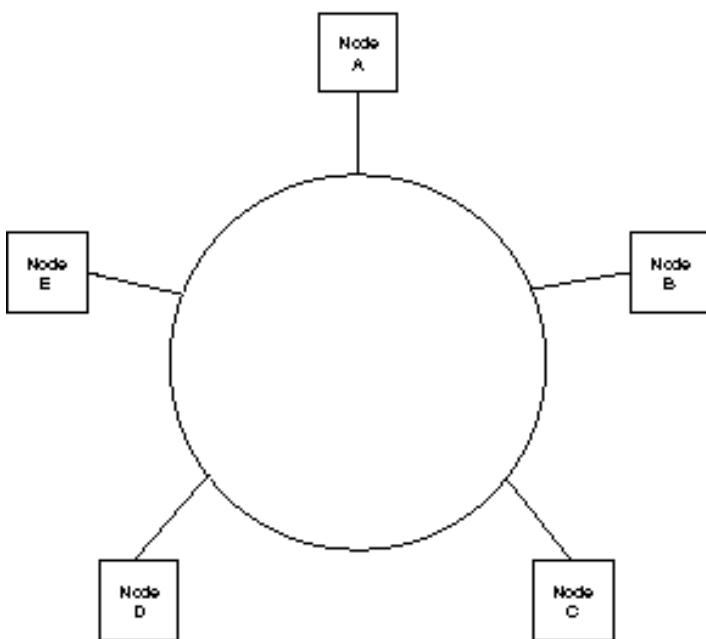
As you see in Chapter 38, “Tuning the Network Components,” there are ways to avoid collisions and increase the performance of an Ethernet network by segmenting the network and reducing overhead.

Token Ring

A Token Ring network differs from an Ethernet network in the basic way the NIC gets access to the network. In an Ethernet network, each NIC simply checks to see whether the network is available and starts transmitting. The Token Ring network uses a more orderly approach. In a Token Ring network, a *token* is sent around the ring, somewhat like a semaphore. A NIC can transfer data only when it has the token. The token is sent around the ring in a circular fashion (see Figure 37.3).

In the Token Ring, just as with Ethernet, the NIC sends the data in a structure called a *packet*. These packets contain not only the data network protocol information but also information concerning the NIC that sent the packet and the receiving NIC’s address. When the device driver sends a packet to the NIC to be sent across the network, the NIC waits until the token is available and then transmits the data.

In a Token Ring network, the structure passed around the network is called a *token*. This token is continually passed around the ring in a circular fashion. If the NIC that gets the token does not need to transmit at that time, it quickly sends the token on to the next NIC in the ring. The token-passing approach is efficient because it prevents collisions.

Figure 37.3*A Token Ring network.*

Token Ring networks are very popular and exhibit very good performance but they are sometimes difficult to manage. Because the token must be passed to each of the machines in the ring, it is necessary to keep the size of each ring as small as possible. Because it is very difficult to maintain a large number of machines on a single ring, there is a need for multiple rings.

Token Ring networks have a bandwidth capability of 16 megabits/second. As you know, in Ethernet, it is very difficult to actually achieve a throughput near the limits of the medium; with Token Ring, however, it is possible to get very near that limit. The token-passing method eliminates the chance of collision and allows you to run at or near maximum throughput.

Fiber Optics

Fiber optics network controllers have been on the market for several years and have become quite popular when high-speed networking is a must. Fiber optics (also known by the mysterious acronym FDDI) controllers operate in a manner very similar to that of Ethernet controllers. Fiber optics networks have a bandwidth of 100 megabits/second.

In a manner similar to Ethernet, the FDDI controller creates an FDDI packet that contains a *network packet* generated by the network protocol driver. Additional FDDI information is placed around the network packet to create the FDDI packet. The structure of the packet is determined by the network protocol being used.

Just as with Ethernet, in an FDDI network, it is possible to get both deferred packets and collisions. You must take the same kind of precautions to avoid these conditions. Just as with Ethernet, it may be necessary to divide your FDDI network functions into separate segments.

As the network is driven closer to its maximum bandwidth, the number of collisions increases exponentially. The more traffic there is on the network, the more likely your packet will collide with another packet. Even though your network may be able to theoretically handle 100 megabits/second, it is unlikely that you will ever achieve this. These things make collisions more likely:

- ◆ **High numbers of connections.** If there are many NICs on the LAN segment, you are more likely to see collisions because it is more likely that a NIC will be transmitting.
- ◆ **Small packet sizes.** If the packet size is small, there will more likely be more packets, and the collision rate will increase.

With smaller packets, there is likely to be more wasted time between packets (that is, time during which there is no network activity). With larger packets, there is less inactivity on the network. Larger packets are more efficient and perform better.

As you see in Chapter 38, “Tuning the Network Components,” there are ways to avoid collisions and increase the performance of an Ethernet network by segmenting the network and reducing overhead.

Other Technologies

As every other aspect of the computer industry, networking is changing and improving rapidly. Other new technologies being introduced or on the horizon include such things as ATM, HPPI (High Performance Parallel Interface), and Fibre Channel networks.

ATM has been very well received and promises high bandwidth rates in several modes. The theoretical bandwidths of ATM are approximately 600 megabits/second and approximately 2.5 gigabits/second in different modes. The ATM networks will gain in popularity in the near future.

HPPI and Fibre Channel are both emerging standards that in the future may provide for even higher performance networks. These standards are currently under development and may take years to finally make it to the mainstream computer network.

Network Protocols

Several very popular network protocols are in use today, including TCP/IP, SPX/IPX, Banyan Vines, AppleTalk, and NetBIOS. This chapter focuses on the two most-popular network protocols: TCP/IP and SPX/IPX. TCP/IP has been in use since the early days of computer networking and is probably the most popular of the network protocols.

The network protocol is responsible for the network communication between the different nodes in the network. The network protocol defines how the messages are sent, who receives them, and how or if the data is routed to other machines or networks. The network protocol is also the determining factor about what other applications or protocols can be layered on top of it.

TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is the most popular network protocol in use today; it runs on virtually every computer system and operating system. As do the hardware devices, the network protocol wraps the data within a structure called an *IP packet*. This packet contains information such as the machine's IP address, the address of the machine the packet is being sent to, and routing information.

TCP/IP has an addressing protocol used to facilitate routing of IP packets. The address is in the form `www.xxx.yyy.zzz` and is a standard. Typically, the first three sets of numbers `www.xxx.yyy` are used to identify the subnet on which the machine is located; the final numbers `zzz` are used to uniquely identify the computer system. Each of the sets of numbers is 8 bits and is known as an *octet*.

For machines on the Internet, the first two octets are registered and are unique to your corporation. The last two octets are available for your own use. Having two octets allows you to divide your network into subnets. A *subnet* is an individual isolated portion of your network.

SPX/IPX

SPX/IPX is a network protocol designed by Novell to support the NetWare operating system. Because NetWare was originally designed around file and print services, it required a high-speed network subsystem. As such, the SPX/IPX protocols are very lightweight and high performance.

Even though it was originally designed for file and print services, SPX/IPX works very well with SQL*Net. Not only does it provide compatibility with the NetWare server, it is a high-performance network transfer protocol. If you operate in a NetWare network, you can easily take advantage of SPX/IPX by using Oracle and SQL*Net as well as NetWare directory services.

Summary

As you have seen, the performance of the network can be affected by hardware and software inefficiencies as well as the load on the network. Probably the most common problem is the one caused by overloading the network. With every type of network, only a certain amount of data can be simultaneously transferred before the network is overloaded.

When an Ethernet network gets extremely busy, you see a large number of collisions. These collisions cause the network to retransmit the data, wasting CPU and network cycles. As you get closer to the limits of the network, you generate more and more collisions.

When you add traffic to a Token Ring network, you see a slowdown caused by the amount of time it takes for the token to return to your system. The more machines on the network that need to transmit data, the more delays you see before it is your turn again.

In summary, the act of exceeding a fixed load causes most network performance problems. The next chapter looks at some of the ways you can avoid or reduce these problems.

Chapter 38

Tuning the Network Components

The last chapter reviewed the basics of how the various network components operate; this chapter looks at how to tune these components. The chapter considers both the hardware and software and how to optimally design a computer network. Once you have quantified the limitations of the network, you can design a network that will function within those limits.

As you saw in the last chapter, the primary cause of network performance problems is because the bandwidth of the network itself has been exceeded. The network itself can support only a certain amount of traffic. When that amount has been reached, the network experiences delays. In turn, these delays can cause noticeable increases in response times.

There are few optimizations you can do to the OS to tune performance, but these fall into the category of fine tuning. There is really not very much you can do to the OS to overcome the limitations of the network itself.

NOTE: When I talk about the limitations of the network, I refer to the actual limitations of the medium you are using. If you use traditional Ethernet, this limitation is 10 megabits/second. If you use 100Base-T, the limitation is 100 megabits/second.

The network limits are fixed by software and hardware standards and the limitations of the hardware. There is nothing you can do to increase the limits of these mediums. By knowing your limitations, you can design your network to fit within the bounds. When the application and database need data faster than the medium can supply it, the network becomes a bottleneck.

The next section starts by looking at some of the tuning you can do to the network software. The following sections provide some design hints that can help you put together a high-performance network. By properly designing the network segments to handle the requested load, you can enhance the performance of the system.

Software Tuning

Several things can be done within the operating system to improve performance. The most important of these are to reduce system overhead caused by networking and to use a sufficiently large packet size to increase throughput. Depending on your operating system, you may or may not be able to change anything to increase performance. The following sections review a few of the network parameters that can be modified on a per-operating system basis.

NetWare

There are a few networking parameters you can modify to increase NetWare's efficiency. By making sure that you have a sufficient amount of packet receive buffers, you can avoid dynamically allocating them at run time. The packet receive buffers are used to buffer the incoming network requests while the processor is busy. Do this by making the NetWare parameter **MINIMUM PACKET RECEIVE BUFFERS** sufficiently high. This parameter is set in STARTUP.NCF. A good starting point is 200 for 40 users or fewer.

Monitor this change on the NetWare monitor screen to see whether the buffers are being dynamically allocated. If the number of packet receive buffers frequently exceeds the value you have set for **MINIMUM PACKET RECEIVE BUFFERS**, think about increasing this value.

If you never achieve more than the minimum, you may be wasting space. If the value is too high, you waste memory. Try setting the value down until you begin to see dynamic allocation at run time. At this point, add 2 to 5 percent and monitor the change on a regular basis. If dynamic allocation starts up again, increase MINIMUM PACKET RECEIVE BUFFERS until dynamic allocation stops.

Windows NT

Although there is little network tuning you can do with Windows NT, there are a few things you can do to streamline the network. By prioritizing the network bindings, you can increase performance. Do this with the Control Panel Network Configuration utility. Put the network protocol you use most frequently first in the list to give it the highest priority; follow it with the next protocol you use, and so on.

Remove any protocols from the list that you do not use; this benefits performance by reducing both memory consumption and CPU overhead. Also be sure that the system is configured as a *server* on the Network Configuration screen.

OS/2

There is very little you can do with OS/2 to tune the network. By installing only the network protocols you will be using, you can streamline the network subsystem. By removing any unneeded components, you also benefit. By removing any unneeded protocols, you save both memory and CPU utilization.

UNIX

With UNIX, there is also very little you can tune to improve network efficiency. The only parameters you may have to adjust relate to increasing the available number of connections and user resources. These parameters are covered in Chapter 12, “Operating System-Specific Tuning.”

As with the other operating systems, you can streamline the system by removing any protocols or network applications that are not necessary for the function of the system. By removing any unnecessary protocols, you save memory and CPU usage.

Oracle Tuning

The best way to improve network performance in software is to design your application to eliminate the transmission of unwanted and unnecessary data. By using stored procedures, you not only reduce the amount of data transmitted but also improve performance by taking better advantage of the shared SQL areas.

Only retrieve useful data from your stored procedures and queries. Any unnecessary data transmitted from the server to the client wastes valuable network bandwidth.

Another way to reduce network traffic within Oracle is to take advantage of local tables. If part of your data is never updated and is heavily used locally, you may see a benefit from replicating the data to your client machine. The replicated data not only reduces network traffic, it speeds access to the data as well.

Optimize the amount of data transmitted across the network as much as possible. The network subsystem in the OS should also be streamlined to provide optimal performance. If you still have a network bottleneck, you have no alternative other than increasing the bandwidth of the network.

Network Design

As you have seen, there are only a few things you can do in the operating system and the application to improve network efficiency. The majority of the performance improvements are accomplished by reducing the load on any particular component in the network. Primarily, you can improve performance by *subnetting*, that is, by segmenting the load among several different segments.

The alternative to increasing the bandwidth of the network is to purchase faster network hardware. By upgrading from 10Base-T to 100Base-T, you instantaneously see substantial improvement in bandwidth. In many cases, your network wiring may already be able to handle the faster 100Base-T networks.

You should use some sort of network load monitor on a regular basis to determine whether you are near the limits of the network. Do some trend analysis and try to determine where the majority of network traffic is originating. You may discover that certain operations (such as backups or loads) are using a large percentage of the network bandwidth and should be segmented off the main network.

If you keep the network usage within limits, the network should not be a bottleneck. As mentioned in the last chapter, as you near the limits of an Ethernet or FDDI network, the number of collisions increases exponentially. It is a good idea to segment these types of networks so that you never exceed 60 to 70 percent of the available bandwidth.

Bandwidth Considerations

The *bandwidth* of the network represents the theoretical maximum of data throughput that can be achieved by the network medium. For Ethernet, this is 10 megabits/second. By using larger packets, you can achieve a maximum throughput much closer to the theoretical limit

than you can with small packets. The more computers that try to access the network, the less likely you are to reach the maximum throughput because of the increased likelihood of collisions.

When your network load reaches 60 to 70 percent of the bandwidth of the network, you will begin to see a fairly high collision rate. At this point, the performance of the network starts to be affected. If possible, try to keep the network usage below 60 percent. Of course, there are peak times during which the network sees high usage, but in general, try to keep the load in this range.

The only available way to increase the bandwidth of the network is by upgrading to a faster type of network hardware, such as 100Base-T, fiber optics, or ATM. The available bandwidth of the network is based on the type of hardware you are using. The speed of the LAN segment is based on standards. By standardizing the speeds, you can add different hardware components supplied by different hardware vendors and they will all work on your network.

Segmenting the Network

The best way to reduce the amount of traffic on your network is to use subnetting or segmenting. Subnetting involves breaking the network into smaller individual LAN segments. By breaking the LAN into smaller segments, you divide the network load among the segments and reduce the load on each individual segment.

By reducing the load on the individual segments, you reduce collisions and avoid performance degradation. When the network is segmented, you may have to deploy routers. Routers are used to pass network packets between different subnets.

Although most operating systems can do routing, doing so uses some CPU resources. You should avoid routing on your database server. Hardware routers are also available; these routers bypass the operating system and can perform routing tasks with very little overhead. Hardware routers can route network packets much faster and more efficiently than any operating system.

Bridges, Routers, and Hubs

Bridges, routers, and hubs are all needed to keep a network running. *Routers* are used to route data back and forth between different subnets; *bridges* are used to bridge between networks; and *hubs* are used simply as electrical repeaters. All these elements may be in use in your installation.

Be sure that your routers and bridges are not a source of delays. Especially if you employ a router or a bridge that uses a general-purpose OS for routing and bridging functions, you may see substantial delays in these services. A special-purpose hardware router or bridge always provides significant performance gains over an OS-based router or bridge.

Summary

To optimize your network, you should look at both the software and the hardware. To optimize the network software, there is typically not much you can do. By reducing unnecessary network protocols and applications, you can streamline the operation of the network; there is not much else you can do to enhance its performance.

To optimize the network hardware, there is not very much you can do either. The best thing to do is simply monitor the network load and break the network into smaller segments if necessary. By subnetting the network, you can reduce the load for each segment, thus avoiding network bottlenecks.

Part

VII

References

A Appendix A Review of Tuning Guidelines

- B** Quick Reference
- C** Flowcharts
- D** Glossary
- E** Oracle Tuning Parameters
- F** Contents of the CD-ROM

Appendix

A

Review of Tuning Guidelines

This appendix is intended as a quick reference to many of the topics presented throughout this book.

RDBMS Tuning

The following sections review many of the concepts presented in Part II, “Tuning the Server.” They review some of the areas of importance and how to find and overcome performance bottlenecks. The following sections relate primarily to the tuning parameters for the OS and Oracle.

SGA

The System Global Area (SGA) contains the shared pool, the redo log buffer, and the database block buffers.

Shared Pool

The shared pool contains the library cache, the data dictionary cache, and the shared session area (with the multithreaded server).

Library Cache

The library cache contains the shared SQL and PL/SQL areas. You can improve performance by both increasing the cache-hit rate in the library cache and by speeding access to the library cache by holding infrequently used SQL statements in cache longer.

The V\$LIBRARYCACHE table contains statistics about how well you are using the library cache. The important columns to view in this table are PINS and RELOADS:

- ◆ *PINS*: The number of times the item in the library cache was executed.
- ◆ *RELOADS*: The number of times the library cache missed and the library object was reloaded.

A low number of reloads relative to the number of executions indicates a high cache-hit rate.

Data Dictionary Cache

The data dictionary cache contains a set of tables and views that Oracle uses as a reference to the database. It is here that Oracle stores information about both the logical and physical structure of the database.

To check the efficiency of the data dictionary cache, check the cache-hit rate. Statistics for the data dictionary cache are stored in the dynamic performance table V\$ROWCACHE (the data dictionary cache is sometimes known as the *row cache*). The important columns to view in this table are GETS and GETMISSES:

- ◆ *GETS*: The total number of requests for the particular item.
- ◆ *GETMISSES*: The total number of requests resulting in cache misses.

A low number of cache misses are expected, especially during startup, when the cache has not been populated.

Shared Session Information

In a multithreaded server configuration, the session information is also stored in the shared pool. This information includes the private SQL areas as well as sort areas. It is important to make sure that you do not run out of memory for this shared session information.

To determine whether you need to increase space for these shared sessions, you can extract the sum of memory allocated for all sessions and the maximum amount of memory allocated for sessions from the dynamic performance table V\$SESSTAT. If the maximum amount of memory used is high, it may be necessary to increase the size of the shared pool. Because the shared pool is used for other functions as well (such as the library cache and the data dictionary cache), it is a good idea to increase the size of the shared pool to accommodate the additional memory usage. If you have enough memory in your system, increase the shared pool by the maximum amount of memory used by the shared server processes. If you have a limited amount of memory, use the sum of memory allocated to sessions (obtained when an average number of users were connected and running) as the basis for the amount of memory for the shared pool.

Database Block Buffer Cache

Probably the most important Oracle cache in the system is the buffer cache. The buffer cache makes up the majority of the Oracle SGA and is used for every query and update in the system.

The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The important columns to view in this table are listed here:

- ◆ *PHYSICAL READS*: The total number of requests that result in a disk access. This is a cache miss.
- ◆ *DB BLOCK GETS*: The number of requests for blocks in current mode.
- ◆ *CONSISTENT GETS*: The number of requests for blocks in consistent mode. Buffers are typically retrieved in consistent mode for queries.

The sum of the values in DB BLOCK GETS and CONSISTENT GETS represents the total number of requests for data.

The cache-hit ratio is determined using this formula:

```
Cache Hit Ratio = 1 - ( PHYSICAL READS / ( DB BLOCK GETS + CONSISTENT GETS ) )
```

The block buffers are the most important area of the SGA and must be tuned because of the large effect they have on the system and the amount of resources they consume.

Performance Enhancements

I prefer to separate the performance-enhancement options from the general tuning of Oracle. Performance enhancements tend to be things that may or may not help your configuration and application; in fact, they may sometimes hurt if you're not careful. On the other hand, tuning parameters always helps, based on correct interpretation of Oracle statistics. The following sections review a few of the enhancements you have seen throughout the book.

Block Size

Depending on your configuration and data access patterns, you may be able to benefit from using a larger block size. With a larger block size, under certain circumstances, you get the benefit of less wasted space and more efficient I/O. Here are a few guidelines that may help you decide whether changing the size of `DB_BLOCK_SIZE` can benefit you:

- ◆ OLTP systems benefit from smaller blocks. If your application is OLTP in nature, you will not benefit from larger blocks. OLTP data typically fits well in the default block size; larger blocks would unnecessarily eject blocks from the SGA.
- ◆ DSS systems benefit from larger blocks. In the DSS system in which table scans are common, retrieving more data at a time results in a performance increase.
- ◆ Larger databases benefit from larger blocks. Larger databases see a space benefit from less wastage per block.
- ◆ Databases with large rows will benefit from larger blocks. If your rows are extremely large (such as images or text) and don't fit in the default block, you will see a definite benefit from a larger block size.

Because changing the block size unnecessarily causes increased I/O overhead, this change *does* carry some risk. Change the block size with caution.

Clusters

A *cluster*, sometimes called an *index cluster*, is an optional method of storing tables in an Oracle database. Within a cluster, multiple related tables are stored together to improve access time to the related items. Clusters are useful in cases where related data is often accessed together. The existence of a cluster is transparent to users and to applications; the cluster affects only how data is stored.

A cluster can be useful for tables in which data is primarily accessed together in a join. In such situations, the reduced I/O needed to bring the additional data into the SGA and the fact that the data is already cached can be a big advantage.

However, for situations in which the tables have a large number of `INSERT` statements or the data is not frequently accessed together, a cluster is not useful and should not be used.

Do not cluster tables if full-table scans are often performed on only one of the tables in the cluster. The additional space required by the cluster and the additional I/O reduces performance.

Reduce Fragmentation

Fragmentation is the condition that occurs when pieces of the database are no longer contiguous. Fragmentation can consist of *disk fragmentation* or *tablespace fragmentation*. Both of these types of fragmentation usually affect performance.

Disk fragmentation usually causes multiple I/Os to occur when one I/O would have been sufficient (for example, with chained or migrated rows). Disk fragmentation can also be caused when the extents that make up the database segments are noncontiguous, which can happen with excessive dynamic growth.

Tablespace fragmentation is caused by the dropping and creation of segments. This can create large free areas between segments, which result in the inefficient use of space and excessive disk seeks over the empty areas. Tablespace fragmentation can also prevent Oracle from taking advantage of multiblock reads.

One way to eliminate fragmentation is to export the table or tablespace data, remove and re-create the table or tablespace, and import the data. By eliminating fragmentation, you can reduce excessive I/Os and CPU usage, streamlining data access. Any overhead and unnecessary I/Os you can reduce will improve system performance.

Hash Clusters

A hash cluster is similar to a cluster except that it uses a hash function rather than an index to reference the cluster key. A hash cluster stores the data based on the result of a hash function. The hash function is a numeric function that determines the data block in the cluster based on the value of the cluster key.

To achieve good performance from a hash cluster, you must meet the following criteria:

- ◆ The cluster key value is unique
- ◆ The majority of queries are equality queries on the cluster key
- ◆ The size of the table is static (very little growth occurs)
- ◆ The value of the cluster key does not change

If you can take advantage of hashing by meeting these strict criteria, you will see very good performance. Hashing is extremely efficient under the right conditions; however, having a hash cluster under the wrong conditions can cause some performance degradation.

Indexes

An index, like the index in this book, is an optional structure designed to help you achieve faster access to your data. When optimally configured and used, indexes can significantly reduce I/O to the data files and greatly improve performance. You must first decide whether an index is appropriate for the data and access patterns in your particular system. Having decided to use an index, you must decide which columns to index. Indexing appropriately can greatly improve performance by reducing I/Os and speeding access times.

Careful planning and periodic testing with SQL Trace can lead to very effective use of indexes, with optimal performance being the outcome.

Multiblock Reads

When performing table scans, Oracle has the capability to read more than one block at a time, thus speeding I/Os. By reading more than one block at a time, a larger chunk of data can be read from the disk, eliminating some disk seeks. By reducing disk seeks and reading larger blocks, both I/O and CPU overhead are reduced.

The amount of data read in a multiblock read is specified by the Oracle initialization parameter `DB_FILE_MULTIBLOCK_READ_COUNT`. The value for this parameter should always be set high because there is rarely any disadvantage in doing so. The size of the individual I/O requests depends on both `DB_FILE_MULTIBLOCK_READ_COUNT` and `DB_BLOCK_SIZE`. A good value for multiblock reads is 64K.

Multiblock Writes

Multiblock writes are similar in nature to multiblock reads and have many of the same requirements. The multiblock write feature is new with Oracle version 7.3. Under certain conditions, you can now perform multiblock writes.

Multiblock writes are available through the direct path loader as well as through sorts and index creations. As with multiblock reads, multiblock writes reduce I/O and CPU overhead by writing multiple database blocks in one larger I/O operation.

The amount of data written in a multiblock write is specified by the Oracle initialization parameter `DB_FILE_MULTIBLOCK_WRITE_COUNT`. The size of the individual I/O requests depends on both `DB_FILE_MULTIBLOCK_WRITE_COUNT` and `DB_BLOCK_SIZE`. As with multiblock reads, a good value is 64K.

Oracle Parallel Query Option

The Oracle Parallel Query option makes it possible for some Oracle functions to be processed by multiple server processes. The functions affected are queries, index creation, data loading, and recovery. For each of these functions, the general principle is the same: Keep the processing going while Oracle is waiting for I/O.

For most queries, the time spent waiting for the data to be retrieved from disk usually overshadows the amount of time actually spent processing the results. With the Parallel Query option, you can compensate for this “wasted time” by using several server processes to execute the query. While one process is waiting for I/Os to complete, other processes can execute. If you are running on a Symmetric Multiprocessor (SMP) computer, a cluster, or a Massively Parallel Processing (MPP) machine, you can take maximum advantage of the Parallel Query option.

The amount of parallelism can be tuned with several of the Oracle initialization parameters:

<i>Parameter</i>	<i>Description</i>
PARALLEL_DEFAULT_MAX_SCANS	Specifies the maximum number of query servers to be used by default for a query. This value is used <i>only</i> if no value is specified in a PARALLEL hint or in the PARALLEL definition clause. This parameter limits the number of query servers used by default when the value of PARALLEL_DEFAULT_SCAN_SIZE is used by the query coordinator.
PARALLEL_DEFAULT_SCAN_SIZE	Specifies the number of query servers to be used for a particular table. The size of the table divided by PARALLEL_DEFAULT_SCAN_SIZE determines the number of query servers, up to PARALLEL_DEFAULT_MAX_SCANS.
PARALLEL_MAX_SERVERS	Maximum number of query servers or parallel recovery processes available for this instance.
RECOVERY_PARALLELISM	The number of processes to be used for instance or media recovery. A large value can greatly reduce instance recovery time. A value of zero or 1 indicates that parallel recovery will not be done and that recovery will be serial. A good value for this parameter is in the range of the number of disks you have (up to 50).

I am a real fan of the Parallel Query option. I have seen great improvements from the use of parallel queries as well as dramatic reductions in recovery time when the parallel recovery feature is used.

Oracle Parallel Server Option

The Oracle Parallel Server option is one of the most innovative and impressive options available from Oracle. With the Parallel Server option, you can cluster several computers together using a shared-disk subsystem and have multiple Oracle instances access the same database. If your application is suitable, you can see very good scalability from adding additional computers.

The Oracle Parallel Server option uses a sophisticated locking mechanism in conjunction with a shared-disk subsystem to allow multiple instances to access the same data. If you have an application that can take advantage of the Oracle Parallel Server architecture, you should see some very good performance improvements.

The two areas that can most influence the performance of your parallel server system are data partitioning and PCM lock management. Both of these can make a huge difference in the performance of your system.

- ◆ **Partitioning.** By properly partitioning your data to reduce lock traffic and contention for blocks between servers, you can enhance performance. Try to balance your users so that the users accessing the same tables are on the same machine; doing so can reduce contention for locks.
- ◆ **PCM locks.** By carefully managing the number of locks on each table, you can enhance performance. Tables with a lot of traffic between nodes should have more locks than tables with less contention. By balancing the number of locks, you can reduce overhead.

TIP: By taking advantage of read-only tablespaces when applicable, you can reduce the number of PCM locks in your system. Because read-only tablespaces do not allow updates, no locking is necessary.

Spin Counts

Multiprocessor environments may benefit by tuning the parameter `SPIN_COUNT`. Under normal circumstances, if a latch is not available, the process sleeps for a while and then wakes up to try the latch again. If you are on a multiprocessor system, it is likely that the process holding the latch is currently processing on another CPU and will be finished in a short time. By setting `SPIN_COUNT` to a value greater than zero, the process spins while counting down from `SPIN_COUNT` to zero. If the latch is still not available, the process goes to sleep.

Setting `SPIN_COUNT` can hurt performance if you're not careful. This parameter should be set only for multiprocessor computers and should be monitored for effectiveness. A good value to try is 2000. The value of `SPIN_COUNT` specifies how many times the process will spin before putting itself to sleep. Because the speed of processors varies, the time it takes to spin also varies, but the speed of the other process holding the desired resource also vary with the speed of the processor.

OS Tuning

OS tuning is very specific and dependent on the OS you are running. There are, however, some general guidelines you can follow.

OS Tuning Goals

The general goal in tuning the server operating system is to simply provide the resources Oracle needs to function optimally. Here's a list of several areas that need attention:

<i>OS Area</i>	<i>Comments</i>
Memory	Enough memory should be allocated to Oracle to hold the entire SGA in physical memory. If your SGA pages out, performance is severely degraded. Paging out of user processes should also be avoided.
I/O	The I/O subsystem should be tuned to have the least amount of overhead possible. By reducing overhead, the performance of the entire system is increased.
Processes	The system should be configured to handle the required number of processes necessary to support your users.

These and other OS features should be used with the goal of reducing system overhead. Any extra CPU time spent in the OS is CPU time not spent processing Oracle.

OS Features

You should make use of any OS feature that can reduce overhead. Although these features vary from platform to platform and are OS specific, some common areas are listed here:

<i>Feature</i>	<i>Comment</i>
Post-wait semaphore	This feature is available on some platforms and reduces some of the overhead associated with traditional semaphores and scheduling.
Scheduling parameters	Some OSes allow you to adjust scheduling to reduce the amount of preemption in the system. If available, use this feature.
Cache affinity	Cache affinity is available in some operating systems and should be used with caution. Although some application such as decision support can benefit, OLTP may suffer slightly from the effects of cache affinity.
Asynchronous I/O (AIO)	Most operating systems have AIO available; you should use it. The use of AIO reduces I/O processing overhead.

All these OS features were designed to reduce excess overhead that takes away from user processing. By optimizing your OS to reduce overhead, you can enhance total system throughput.

I/O Tuning

The I/O system should be designed and implemented with the following goals in mind:

- ◆ **Isolate sequential I/O.** By isolating sequential I/O so that it is purely sequential to the disk, you can greatly enhance throughput. Any random I/O to these disks degrades performance. Writes to the redo log are sequential.
- ◆ **Spread out random I/O.** Random I/O performance can be increased by adding more disk drives to the system and spreading out the I/Os among the disks. I/Os to the data files are typically random (especially in OLTP systems).

By following these guidelines and planning your system so that your disk drives can support the amount of disk I/O demanded of them, I/O should not be a problem.

System Design

Part III, “Configuring the System,” looked at several types of systems and determined the data access patterns, the system load, and optional features that can enhance performance. The systems that were reviewed are listed here:

<i>System</i>	<i>Description</i>
OLTP	Characteristics include many users, high I/O rates, much random I/O, and response-time constraints. The Oracle Parallel Server option is possibly a win for OLTP systems.
Batch	Characteristics include system load-time constraints, index build-time constraints, and long-running queries. The Oracle Parallel Query option can benefit the typical batch system. Large block sizes also help.
DSS	Characteristics include system load-time constraints, index build-time constraints, and long-running queries. The Oracle Parallel Query option definitely benefits the typical DSS system. Large block sizes and multiblock reads also help. The Oracle Parallel Server option may also benefit the DSS system.
Data warehouse	The characteristics of a data warehouse system are much like those of a super DSS system: a huge amount of data, large queries, and heavy I/O usage. Both the Parallel Query and Parallel Server options may benefit the data warehouse system.

<i>System</i>	<i>Description</i>
BLOBs	BLOBs require significant I/O and response time. Data-feed interruption cannot be tolerated. Large block sizes are a must.
Parallel server	The Oracle parallel server system can be enhanced by carefully partitioning tasks and allocating PCM locks efficiently. Because the parallel server system involves two or more systems working together, balancing transactions and functions is crucial.
Optimal backup	If backup and recovery time is critical, there are several ways your system can be configured to optimize for these tasks.
Miscellaneous	Other Oracle products (Oracle Financials, Oracle WebSystem, and others) have their own unique characteristics and types of enhancements.

By looking at various types of systems and examining their characteristics, Part III, “Configuring the System,” gave you a feel for how these systems operate. By understanding the systems, you have a better idea of how to tune them for optimal performance.

Application Tuning

Part IV, “Tuning SQL,” looked at tuning the application, which mainly involves tuning the SQL statements themselves. Several chapters explained how to tune specific SQL statements and how to use Oracle features to your advantage. Here is a list of some of the important topics discussed in Part IV:

- ◆ **Using EXPLAIN PLAN.** By using EXPLAIN PLAN, you can analyze the execution plan the optimizer has chosen for the SQL statement.
- ◆ **Using SQL Trace.** By using SQL Trace, you can analyze the execution of the SQL statement and determine any potential bottlenecks.
- ◆ **Tuning SQL statements.** You can improve performance by making SQL statements take advantage of such things as indexes, clusters, and hash clusters.
- ◆ **Using the Oracle optimizer.** You learned how the optimizer works and how you can best take advantage of it.
- ◆ **Using procedures and packages.** You learned how to use procedures, functions, and packages to improve performance.
- ◆ **Providing for data integrity.** You learned about the importance of data integrity and how to optimally provide for it.

- ◆ **Using hints.** You learned how to use hints to take advantage of information you know about the data and application.
- ◆ **Miscellaneous topics.** The last chapter of Part IV provided tips and miscellaneous topics concerning the tuning of SQL statements.

By now, you should have an idea about the importance of tuning your SQL statements. You should also know how to take advantage of various Oracle features to improve the performance of those statements.

Client Tuning

Part V, “Tuning the Client,” looked at the client computer itself. You learned about some of the things that can affect client performance and how to solve those problems. You also looked at how to tune the client system and how to take advantage of some popular graphical development tools. You saw how to use these tools to improve the performance of the SQL statements generated by those tools. Here is a list of some of the key topics in Part V:

- ◆ **Client bottlenecks.** You learned which parts of the client are likely to become bottlenecks: the application, the network, and the client hardware.
- ◆ **Memory.** Make sure that your client machine has enough memory to run the application. Memory is typically the only hardware performance problem on the client machine.
- ◆ **CPU.** As applications become increasingly more robust and feature rich, the need for faster CPUs is also increasing. You may have to upgrade your CPU if it is not fast enough.
- ◆ **Network.** Although network performance is usually not a problem, it can be in some situations. Make sure that your LAN segments are not overloaded.
- ◆ **Use compiled applications.** Some application development tools use an interpretive language. For best performance, use a compiled application.
- ◆ **Use 32-bit applications.** If you are running Windows NT or Windows 95, be sure that you use 32-bit applications; 16-bit applications are not as efficient on these systems.
- ◆ **Use native drivers.** Although using ODBC can make connections to various servers more convenient, it hurts performance. Use native drivers whenever possible.
- ◆ **Optimize SQL.** Even though your application may have been developed by a GUI development tool, you still may have to optimize your SQL statements.

You should now understand what affects the performance of the client machine and how to solve those problems. Part V of this book also introduced middleware products and how they can be used to improve the overall performance of your configuration.

Network Tuning

Part VI, “Tuning the Network,” looked at the capacity of the network and how you can tune it. The real way to tune the physical network is to stay within its capacity. As your needs grow, you must improve the capacity of your network.

Appendix **B**

Quick Reference

This appendix contains quick summaries you can use to refresh your memory on some of the areas covered in this book.

Oracle Instance Tuning

This section reviews Oracle instance tuning: where to look for performance information in the dynamic performance tables as well as a review of tuning guidelines.

Library Cache

The V\$LIBRARYCACHE table contains statistics on how well you are using the library cache. The important columns to view in this table are PINS and RELOADS:

PINS	The number of times the item in the library cache was executed.
RELOADS	The number of times the library cache missed and the library object was reloaded.

Cache Miss Rate = $\text{SUM(Reloads)} / \text{SUM(Pins)}$

SQL Statements

Total Miss Percent:

```
SELECT SUM(reloads) "Cache Misses",
       SUM(pins) "Executions",
  100 * ( SUM(reloads) / SUM(pins) ) "Cache Miss Percent"
FROM v$librarycache;
```

Individual Statistics:

```
SELECT namespace,
       reloads "Cache Misses",
       pins "Executions"
FROM v$librarycache;
```

Data Dictionary Cache

Statistics for the data dictionary cache are stored in the dynamic performance table V\$ROWCACHE (the data dictionary cache is sometimes known as the *row cache*). The important columns to view in this table are GETS and GETMISSES:

GETS	The total number of requests for the particular item.
GETMISSES	The total number of requests resulting in cache misses.

Cache Miss Rate = $\text{SUM(GETMISSES)} / \text{SUM(GETS)}$

SQL Statements

Total Miss Percent:

```
SELECT SUM(getmisses) "Cache Misses",
       SUM(gets) "Requests",
  100 * ( SUM(getmisses) / SUM(gets) ) "Cache Miss Percent"
FROM v$rowcache;
```

Individual Statistics:

```
SELECT parameter,
getmisses "Cache Misses",
gets "Requests"
FROM v$rowcache;
```

Database Buffer Cache

The statistics for the buffer cache are kept in the dynamic performance table V\$SYSSTAT. The important columns to view in this table are PHYSICAL READS, DB BLOCK GETS, and CONSISTENT GETS:

PHYSICAL READS	The total number of requests that result in a disk access. This is a cache miss.
DB BLOCK GETS	The number of requests for blocks in current mode.
CONSISTENT GETS	The number of requests for blocks in consistent mode. A consistent-mode request is one that is satisfied from a rollback record for consistency.

```
Cache Hit Ratio = 1 - PHYSICAL READS / ( DB BLOCK GETS + CONSISTENT GETS )
```

SQL Statement

Block Buffer Information:

```
SELECT name, value
FROM v$sysstat
WHERE name IN ('db block gets', 'consistent gets', 'physical reads');
```

Remember: **DB Block Buffer Size = DB_BLOCK_BUFFERS * DB_BLOCK_SIZE**

Physical I/O Usage

Information about disk accesses is kept in the dynamic performance table V\$FILESTAT. Important information in this table is found in the following columns:

PHYRDS	The number of physical reads done to the data file.
PHYWRTS	The number of physical writes done to the data file.

The information in V\$FILESTAT is referenced by file number. The dynamic performance table V\$DATAFILE contains a reference to this number as well as this useful information:

NAME	The name of the data file.
STATUS	The type of file and its current status.
BYTES	The size of the data file.

SQL Statement

I/O Information:

```
SELECT SUBSTR(name,1,40), phyrds, phywrts, status, bytes
FROM v$datafile df, v$filestat fs
WHERE df.file# = fs.file#;
```

Remember that disks have certain inherent limitations that cannot be exceeded.

Disk I/O Review

Sequential I/O

Data is written to or read from the disk in order; very little head movement occurs. Access to the redo log files is always sequential.

Random I/O

Data is accessed in different places on the disk; lots of head movement occurs. Access to the data files is almost always random. For database loads, access is sequential; in most other cases (especially OLTP), access patterns are almost always random.

Disk I/O Rates

Remember these general limitations (and refer to Chapter 14, “Advanced Disk I/O Concepts”):

Sequential I/O

A typical SCSI-II disk drive can support approximately 100 to 150 sequential I/Os per second.

Random I/O

A typical SCSI-II disk drive can support approximately 50 to 60 random I/Os per second.

I/O Rules of Thumb

Isolate sequential I/Os

Because sequential I/Os can occur at a much higher rate, isolating them lets you run these drives much faster.

Spread out random I/Os

You can accomplish this goal by striping table data using Oracle striping, OS striping, or hardware striping.

Separate data and indexes

By separating a heavily used table from its index, you allow a query to a table to access data and indexes on separate disks simultaneously.

Reduce non-Oracle I/O

Eliminate non-Oracle disk I/O from disks that contain database files. Any other disk I/O slows down Oracle’s access to these disks.

Chained Rows

You can check for chained rows with the ANALYZE command's LIST CHAINED ROWS option. Use these SQL statements to check for chained or migrated rows:

```
Rem Create the Chained Rows Table
Rem
CREATE TABLE chained_rows (
  owner_name    varchar2(30),
  table_name    varchar2(30),
  cluster_name  varchar2(30),
  head_rowid    rowid,
  timestamp      date);
Rem
Rem Analyze the Table in Question
Rem
ANALYZE
  TABLE scott.emp LIST CHAINED ROWS;
Rem
Rem Check the Results
Rem
SELECT * from chained_rows;
```

Recursive Calls

You can check the number of recursive calls through the dynamic performance table V\$SYSSTAT. Use the following command:

```
SELECT name, value
FROM v$SYSSTAT
WHERE name = 'recursive calls';
```

Rollback Segment Contention

You can tell whether you are seeing contention on rollback segments by looking at the dynamic performance table V\$WAITSTAT. V\$WAITSTAT contains the following data related to rollback segments:

UNDO HEADER	The number of waits for buffers containing rollback header blocks.
UNDO BLOCK	The number of waits for buffers containing rollback blocks other than header blocks.
SYSTEM UNDO HEADER	Same as UNDO HEADER for the SYSTEM rollback segment.
SYSTEM UNDO BLOCK	Same as UNDO BLOCK for the SYSTEM rollback segment.

SQL Statement

Rollback Information:

```
SELECT class, count
FROM V$WAITSTAT
WHERE class IN
('undo header', 'undo block', 'system undo header', 'system undo block');
```

Dynamic Rollback Growth

To determine whether dynamic growth of rollback segments is a problem, look in the dynamic performance table V\$ROLLSTAT. The following columns are of particular interest:

EXTENTS	Number of rollback extents.
RSSIZE	The size in bytes of the rollback segment.
OPTSIZE	The size that the OPTIMAL parameter was set to.
AVEACTIVE	The current average size of active extents. Here <i>active extents</i> are extents with uncommitted transaction data.
AVESHINK	The total size of free extents divided by the number of shrinks.
EXTENDS	The number of times the rollback segment added an extent.
SHRINKS	The number of times the rollback segment shrank. This shrink may be one or more extents at a time.
HWMSIZE	The high-water mark of rollback segment size. This is the largest that the segment ever grew to be.

SQL Statement

Rollback Dynamic Growth Information:

```
SELECT substr(name,1,40), extents, rssize, aveactive, aveshrink, extends, shrinks
FROM v$rollname rn, v$rollstat rs
WHERE rn.usn = rs.usn;
```

Redo Log Buffer Contention

Information concerning redo log buffer contention is stored in the dynamic performance table V\$SYSSTAT as the value of the *redo log space requests* entry. If this number is not zero, it means that a process had to wait for space in the redo log buffer; you should increase its size. Check for this condition with the following SQL statement.

SQL Statement

Redo Log Buffer Contention Information:

```
SELECT name, value
FROM v$sysstat
WHERE name = 'redo log space requests';
```

Redo Latch Contention

Latch contention can be determined by examining the dynamic performance table V\$LATCH. The significant values are described here:

GETS	<i>For willing-to-wait requests.</i> The number of successful requests.
MISSES	<i>For willing-to-wait requests.</i> The number of times the initial request failed.
SLEEPS	<i>For willing-to-wait requests.</i> The number of times subsequent requests failed.
IMMEDIATE_GETS	<i>For immediate requests.</i> The number of successful requests.
IMMEDIATE_MISSES	<i>For immediate requests.</i> The number of times the request failed.

SQL Statement

Redo Latch Contention Information:

```
SELECT SUBSTR(name,1,20), gets, misses, sleeps, immediate_gets, immediate_misses
FROM v$latch
WHERE name IN ('redo allocation', 'redo copy');
```

Sort Performance

One way to tell whether your sorts are occurring in memory or on disk is to look in the dynamic performance table V\$SYSSTAT. The statistics of interest are give here:

sorts (memory)	The number of sorts that were able to fit in the in-memory sort area.
sorts (disk)	The number of sorts that required temporary space on disk.

SQL Statement

Sort Performance Information:

```
SELECT name, value
FROM v$sysstat
WHERE name IN ('sorts (memory)', 'sorts (disk)');
```

Free List Contention

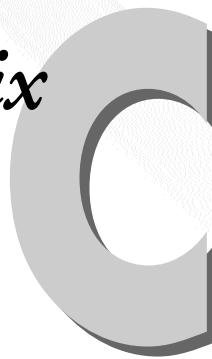
Contention on the free list can be determined by looking at the dynamic performance table V\$WAITSTAT using the following SQL statement.

SQL Statement

Free List Contention Information:

```
SELECT class, count
FROM v$waitstat
WHERE class = 'free list';
```

Appendix



Flowcharts

This appendix contains a few flowcharts to refresh your memory about some of the topics and procedures discussed in this book.

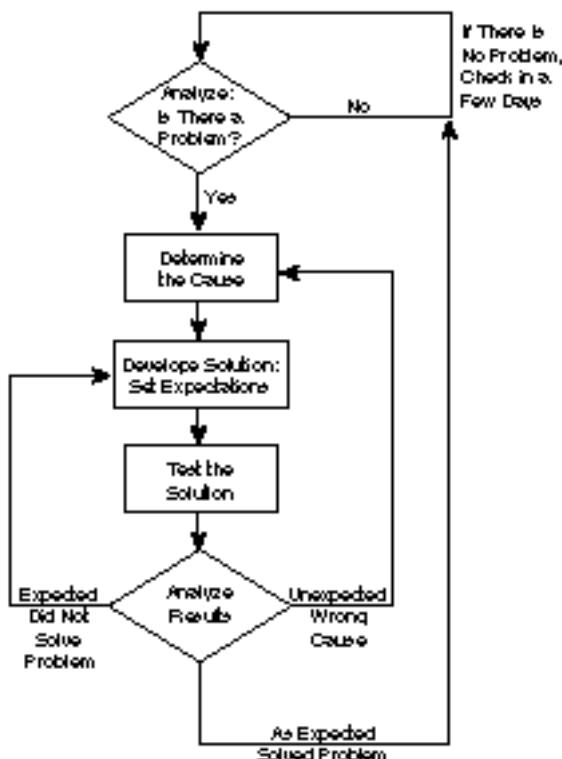
Problem-Solving Methodology

The flowchart in Figure C.1 represents the problem-solving methodology presented in Chapter 4, “Tuning Methodology.” Here are some of the key points:

- | | |
|----------------------|--|
| Analyze the Problem | This step involves understanding what the issues are, if any. You may determine that there is no problem. |
| Determine the Cause | Look into what is causing the problem you are experiencing. It is not always obvious. |
| Determine a Solution | In this step, you determine a course of action and set expectations. It may be that you expect that your solution won’t fix the problem, but determining the solution always provides you with valuable information. |
| Test the Solution | In this step, you test your assumptions and try a fix. |
| Analyze the Result | In this step, you determine the success or failure of your solution. It may be that a failure to fix the problem is the expected result but that it provides good information about the cause of the problem. |

Figure C.1

The problem-solving methodology flowchart.

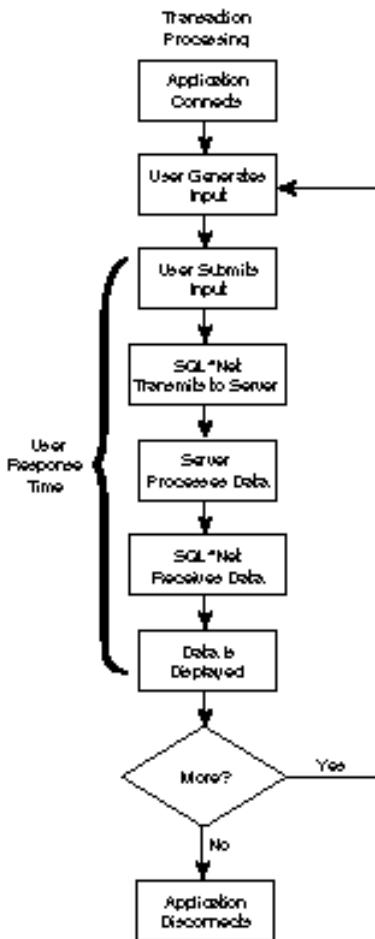


User-Transaction Profile

The flowchart in Figure C.2 represents the user-transaction profile presented in Chapter 24, “What Is a Well-Tuned Statement?”

Figure C.2

The user-transaction profile.



SQL Statement Processing

The flowchart in Figure C.3 is the same flowchart presented in Chapter 24. It shows the steps taken to process SQL statements.

The Oracle Optimizer

The flowchart in Figure C.4 represents the workings of the Oracle optimizer as described in Chapter 27, “Using the Oracle Optimizer.”

Figure C.3
SQL statement processing flowchart.

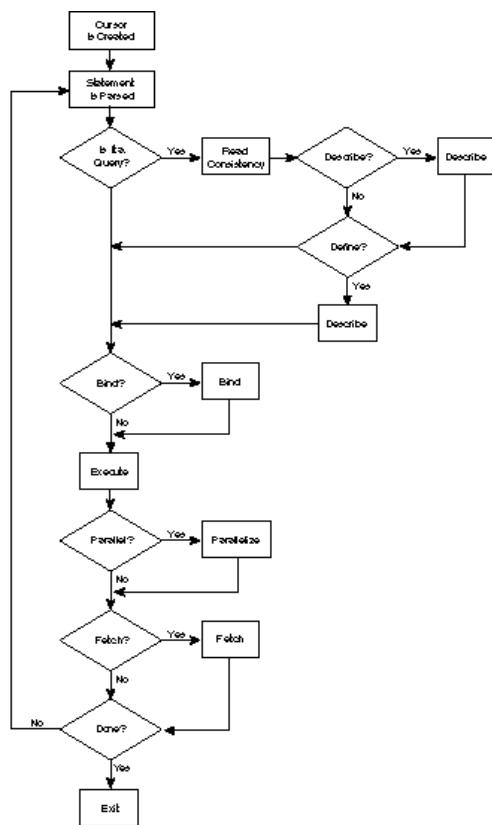
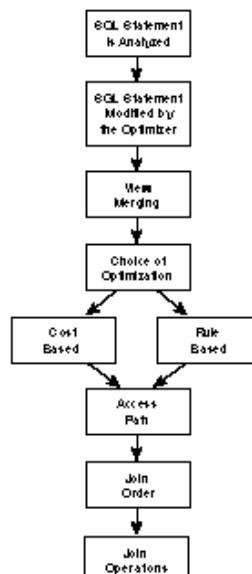


Figure C.4
The Oracle optimizer.

The Oracle Optimizer



Appendix

D

Glossary

I have always thought that a glossary and an index greatly enhance the value of a book. I have made it a priority to have a complete and accurate glossary and index. Enjoy.

ad-hoc. From the Latin *this is*. This term is used to describe an impromptu or spontaneous action. Most commonly used in terms of an *ad-hoc query* to mean an *impromptu, simple query*.

aggregate functions. Functions that operate on the collection of values in a certain column. These operations include such things as SUM, COUNT, AVG, MAX, and so on.

Asynchronous I/O (AIO). Asynchronous I/O allows a process to submit an I/O and not have to wait for the response. Later, when the I/O is completed, an interrupt occurs or the process can check to see whether the I/O has completed. By using Asynchronous I/Os, the DBWR can manage multiple writes at once so that it is not starved waiting for I/Os to complete.

bandwidth. A term often associated with networks or computer busses. The bandwidth is the throughput capacity. The bandwidth of the bus is the maximum rate at which data can be transferred across the bus.

batch processing system. Used to perform large background jobs, usually within a certain specified time window.

BLOB (Binary Large Data). A large amount of binary data stored within an Oracle database. BLOB data can consist of audio, video, images, documents, and so on; it is usually stored as LONG data.

block. The smallest unit of storage in an Oracle database. The database block contains header information concerning the block itself as well as the data.

buffer. An amount of memory used to store data. A buffer stores data that is about to be used or that has just been used. In many cases, buffers are in-memory copies of data that is also on disk. Buffers can be used to hold copies of data for quick read access, they can be modified and written to disk, or they can be created in memory as temporary storage.

In Oracle, the database buffers of the SGA store the most recently used blocks of database data. The set of database block buffers is known as the *database buffer cache*. The buffers used to temporarily store redo entries until they can be written to disk are known as the *redo log buffers*.

A *clean buffer* is a buffer that has not been modified. Because this buffer has not been changed, it is not necessary for the DBWR to write this buffer to disk. A *dirty buffer* is a buffer that has been modified. It is the job of the DBWR to eventually write all dirty block buffers out to disk.

cache. A storage area used to provide fast access to data. In hardware terms, the cache is a small (relative to main RAM) amount of memory that is much faster than main memory. This memory is used to reduce the time it takes to reload frequently used data or instructions into the CPU. CPU chips themselves contain small amounts of memory built in as a cache.

In Oracle, the block buffers and shared pool are considered caches because they are used to store data and instructions for quick access. Caching is very effective in reducing the time it takes to retrieve frequently used data.

Caching usually works using a *least recently used* algorithm. Data that has not been used for a while is eventually released from the cache to make room for new data. If data is requested and is in the cache (a phenomenon called a *cache hit*), the data is retrieved from the cache, avoiding having to retrieve it from memory or disk. Once the data has been accessed again, it is marked as recently used and put on the top of the cache list.

Cartesian products. The result of a join with no join condition. Each row in a table is matched with every row of another table.

checksum. A number calculated from the contents of a storage unit such as a file or data block. Using a mathematical formula, the checksum number is generated from data. Because it is highly unlikely that data corruption can occur in such a way that the checksum would remain the same, checksums are used to verify data integrity. Beginning with Oracle version 7.2, checksums can be enabled on both data blocks and redo blocks.

cluster (machine). A group of computers that together form a larger logical machine. Oracle clusters computers with the Oracle Parallel Server option.

cluster (table). A set of independent tables with a common column stored together. A cluster can improve performance by reducing I/Os and by preloading related data into the SGA before it is needed.

cluster index. The index on the cluster key. Each cluster key must have an index before data can be entered into the cluster.

cluster key. The common column in the set of tables built into a cluster. The cluster key must be indexed.

cold data. This term typically refers to infrequently used data. Cold data is rarely in cache because it is infrequently accessed.

cold database. This term typically refers to a database that is currently closed and not mounted. No users can connect to the database and no data files can be accessed.

collision. Typically refers to a network collision. A network collision occurs when two or more NICs try to use the network at the same time. When this happens, all the NICs must resend their data.

complex statements. An SQL statement that contains a subquery. The *subquery* is a query within the SQL statement used to determine values in the main or *parent* statement.

compound query. A query in which the set operators (`UNION`, `UNION ALL`, `INTERSECT`, and `MINUS`) are used to join two or more simple or complex statements. The individual statements in the compound query are referred to as *component queries*.

concurrency. The ability to perform many functions at the same time. Oracle provides for concurrency by allowing many users to access the database simultaneously.

consistent mode. In this mode, Oracle provides a consistent view of data from a certain point in time for the duration of the transaction. Until the transaction has completed, the data cannot change.

consistent read. A data access in consistent mode.

constraint. The mechanism that ensures that certain conditions relating columns and tables are maintained.

contention. A term usually used to describe a condition that occurs when two or more processes or threads attempt to obtain the same resource. The results of contention can vary depending on the resource in question.

cost-based optimizer. The Oracle optimizer that chooses an execution plan based on information and statistics that it has for tables, indexes, and clusters.

current mode. The mode in which Oracle provides a view as the data exists at this moment. Queries typically use consistent mode.

current read. A read in current mode; typically used for `UPDATE`, `INSERT`, and `DELETE` statements.

cursor. A handle to a specific private SQL area. You can think of a cursor as a pointer to or a name of a particular private SQL area.

data dictionary. A set of tables Oracle uses to maintain information about the database. The data dictionary contains information about tables, indexes, clusters, and so on.

data warehouse. An extremely large database made up of data from many sources to provide an information pool for business queries.

DBA (database administrator). The person responsible for the operation and configuration of the database. The DBA is the person responsible for the performance of the database. The DBA is charged with keeping the database operating smoothly, ensuring that backups are done on a regular basis (and that the backups work), and installing new software. Other responsibilities may include planning for future expansion and disk space needs; creating databases and tablespaces; adding users and maintaining security; and monitoring the database and retuning it as necessary. Large installations may have teams of DBAs to keep the system running smoothly; alternatively, the tasks may be segmented among the DBAs.

DDL (Data Definition Language) commands. The commands used in the creation and modification of schema objects. These commands include the ability to create, alter, and drop objects; grant and revoke privileges and roles; establish auditing options; and add comments to the data dictionary. These commands are all related to the management and administration of the Oracle database. Before and after each DDL statement, Oracle implicitly commits the current transaction.

deadlock. Deadlocks occur when two or more processes hold a resource that the other one needs. Neither of the processes will release its resource until it has received the other's resource; therefore, neither process can proceed.

decision support system. Characterized by large business queries designed to provide valuable data that is used to make sound business decisions.

deferred frame. A network frame delayed from transferring because the network is busy.

DELETE. The SQL statement used to delete a row or rows from a table.

device driver. The piece of software, supplied by the OS vendor or the hardware vendor, that provides support for a piece of hardware such as a disk array controller or a NIC.

disk array. A set of two or more disks that may appear to the system as one large disk. A disk array can be either a software or a hardware device.

DML (Data Manipulation Language) commands. The commands that allow you to query and modify data within existing schema objects. Unlike the DDL commands, a commit is not implicit. DML statements consist of `DELETE`, `INSERT`, `SELECT`, and `UPDATE` statements; `EXPLAIN PLAN` statements; and `LOCK TABLE` statements.

dynamic performance tables. Tables created at instance startup and used to store information about the performance of the instance. This information include connection information, I/Os, initialization parameter values, and so on.

Ethernet. A network hardware standard. Ethernet is probably the most-used network type in the world.

equijoin. A join statement that uses an equivalency operation. The converse of this is the `nonequijoin` operation.

extent. A group of contiguous data blocks allocated for a table, index, or cluster. Extents are added dynamically as needed.

foreign key. An attribute requiring that a value must exist in another object, if not `NULL`, and be its primary key.

frame. See `network frame`.

function. A set of SQL or PL/SQL statements used together to execute a particular function. Procedures and functions are identical except that functions always return a value (procedures do not). By processing the SQL code on the database server, you can reduce the amount of instructions sent across the network and returned from the SQL statements.

HAL (Hardware Abstraction Layer). A software layer closest to the hardware that performs all hardware-specific functions. The HAL layer includes the device drivers.

hot data. This term typically refers to frequently accessed data. Hot data typically gets a good cache-hit rate.

hot database. This term typically refers to a database that is currently mounted, open, and servicing transactions. The instance is up and users are accessing data.

index. A device designed to give you faster access to your data. An index lets you avoid reading through data sequentially in order to find the item you are seeking.

initialization parameter. A parameter read by Oracle at instance startup. These parameters affect the Oracle configuration.

INSERT. The SQL statement used to insert a row into a table.

instance. The Oracle instance is made up of the SGA, the Oracle background processes, and the data files that make up your database.

I/O (Input and Output [of data]). This term can be used to describe any type of data transfer but is typically associated with accesses to disk drives.

join. A query that selects data from more than one table. The data selected from the different tables is determined by conditions specified within the `FROM` clause of the statement. These conditions are called *join conditions*.

join condition. The specification within the `WHERE` clause of a join query that specifies the manner in which the rows in the different tables are paired.

LAN (local area network). A local high-speed network that uses network hardware such as Ethernet or Token Ring and protocols such as TCP/IP and SPX/IPX.

lightweight process. Sometimes known as a *thread*. Similar to a process but shares the process context with other lightweight processes. A lightweight process has much less overhead associated with it than does a normal process. A *thread switch* (change between threads) has much less overhead than a process switch.

logical disk. A term used to describe a disk that is in reality two or more disks in a hardware or software disk array. To the user, it appears as one large disk when in reality it is two or more striped physical disks.

main memory. A term often used to describe RAM (Random Access Memory). This is the part of the computer system used to store data being processed or data that has recently been accessed. RAM is volatile and is not saved when the system is powered off.

microkernel. The core component of a microkernel operating system. The microkernel contains the base components of the operating system. In a microkernel architecture, OS functions usually done in the kernel (such as I/O and device driver support) are moved out of the kernel.

MPP (Massively Parallel Processor) system. A multiprocessor computer consisting of many independent processors that communicate through a complex, high-speed bus.

multiprocessor system. A computer that has two or more CPUs. A multiprocessor can be an SMP (Symmetric Multiprocessor) or an MPP (Massively Parallel Processor) system.

network frame. The structure sent across the network which contains user data as well as network control information. The terms *network frame* and *network packet* are sometimes interchangeable.

network packet. The structure built by the Network Protocol layer. This structure includes user data as well as network and routing information.

NIC (Network Interface Card). A piece of hardware used to network computers together. A NIC can be one of several varieties including Ethernet, Token Ring, or fiber optics.

nonequijoin. A join statement that does not use an equality operation. The converse of this is the equijoin operation.

offline. This term typically refers to a database that is currently closed and not mounted. No users can connect to the database and no data files can be accessed.

OLTP (OnLine Transaction Processing). An OLTP system is characterized by large numbers of users inserting and retrieving data in a somewhat unstructured manner.

online. This term typically refers to a database that is currently mounted, open, and servicing transactions. The instance is up and users are accessing data.

optimizer. A component of the Oracle RDBMS used to select SQL execution plans in the most efficient and cost-effective manner. There are two optimizers: a cost-based optimizer and a rule-based optimizer. Each determines the best execution plan based on different criteria.

Oracle Call Interface (OCI). The standard set of calls used to access the Oracle database.

outer join. A join operation that uses the outer join operator (+) in one of the join statements. The result of an outer join is the rows that satisfy the join condition and those rows in the first table for which no rows in the second table satisfy the join condition.

package. A collection of related, stored procedures or functions grouped together.

packet. See network packet.

paging. An operating system function used to copy virtual memory between physical memory and the paging file (*see* virtual memory). Paging is used when the amount of virtual memory in use has exceeded the amount of physical memory available. Paging is an expensive task in terms of performance and should be avoided if possible.

Parallel Query option. An add-on package to the Oracle RDBMS that allows for concurrent processing of some functions.

Parallel Server option. An add-on package to the Oracle RDBMS that allows for multiple systems to share a common database. Each system has its own instance but the database tables are shared. Data consistency is guaranteed by means of a sophisticated locking mechanism.

physical memory. The actual hardware RAM (Random Access Memory) available in the computer for use by the operating system and applications.

PL/SQL. A set of procedural language extensions that Oracle has added to standard SQL. Procedures, functions, packages, and triggers are written in the PL/SQL language.

primary key. Attributes used to uniquely identify a row in a table.

procedure. A set of SQL or PL/SQL statements used together to execute a particular function. Procedures and functions are identical except that functions always return a value (procedures do not). By processing the SQL code on the database server, you can reduce the amount of instructions sent across the network and returned from the SQL statements.

program unit. In Oracle, the term used to describe a package, a stored procedure, or a sequence.

query. A question. A SELECT statement is considered a query because it requests information from the database. Any read-only SQL statement can be thought of as a query.

random I/O. Occurs when data is accessed on a disk drive in no specific order. Random I/O typically creates significant disk head movement.

read consistency. An attribute used to ensure that, during a SQL statement, data returned from Oracle is consistent. Oracle uses the rollback segments to ensure read consistency.

recursive call. A set of SQL statements generated by Oracle in response to some action or event.

redo log file. The file that contains a copy of all data blocks that have been modified as the result of a database transaction. In the event of a system failure, any transaction can be recovered with these redo blocks. Oracle requires at least two redo log files that are written to in a round-robin fashion.

referential integrity. A constraint on a column in a table that references another column. The constraint can be used to guarantee that the referenced value exists.

replication. The creation of an image of a database or table on another computer system. A *replicated database* is a copy of another database.

rollback. The act of undoing changes that have been made by a transaction.

rollback segment. The place in the database where undo information is kept and can be obtained if a rollback is needed.

rule-based optimizer. The Oracle optimizer that chooses an execution plan based on a table of costs associated with various operations.

scalability. Typically used in association with multiprocessor or cluster configurations. The scalability of the additional component refers to the performance gain obtained by adding that component. A perfectly scaleable solution gives double the performance when you add a second component.

For example, if you have an SMP machine with a measured performance of 1.0 (normalized), add a second CPU, and get a performance of 1.9, the scalability of adding the second CPU is 1.9 or 90 percent. This term is used quite frequently in hardware and software manufacturer's literature when marketing multiprocessor or clustered solutions.

schema. A collection of objects associated with the database.

schema objects. Abstractions or logical structures that refer to database objects or structures.

Schema objects consist of such things as clusters, indexes, packages, sequences, stored procedures, synonyms, tables, views, and so on.

segment. The set of extents that have been allocated to a specific object. Segment types consist of data, index, cluster, hash, and rollback types.

self join. A join in which a table is joined with itself.

sequences. A convenience feature of Oracle that allows unique sequential numbers to be automatically generated for you.

sequential I/O. Occurs when data is accessed on a disk drive in order. Sequential I/O typically causes very little disk head movement.

Server Manager. Oracle's GUI database administration tool. Server Manager is used to replace SQL*DBA.

session. The set of events that occurs from when a user connects to the Oracle RDBMS to when that user disconnects.

SGA. See System Global Area.

shared pool. The area in the SGA that contains the data dictionary cache and shared parsed SQL statements.

simple statement. An SQL statement that involves only one `INSERT`, `UPDATE`, or `DELETE` statement.

SMP (Symmetric Multiprocessor). An SMP system is a multiprocessor computer that uses a shared memory architecture. SMP systems are usually either a tightly-coupled or a loosely-coupled architecture.

snapshot. A copy of a database or table. This term is used in relation to database replication.

SPX/IPX. A network protocol developed for the NetWare operating system. Today, SPX/IPX runs on many operating systems.

SQL*DBA. The Oracle database administration tool. SQL*DBA is being made obsolete by Server Manager.

SQL*Loader. The Oracle database loading tool.

SQL*Net. The Oracle component that allows connections from a network into the Oracle RDBMS. SQL*Net supports many protocols; SQL*Net on any architecture can talk to SQL*Net on any other supported architecture.

SQL*Plus. An Oracle-supplied tool that allows users to run SQL statements directly.

streaming. A term usually associated with a tape device. Tapes perform best when the tape is continually in motion, or streaming. If the data is not fed to the tape quickly enough, the tape drive must reposition the tape to wherever it last stopped recording data (to reposition the tape, the drive must stop the tape and rewind it). This action severely degrades performance.

stored function. *See* function.

stored procedure. *See* procedure.

subquery. A SELECT statement referenced in an UPDATE, INSERT, or DELETE statement.

swapping. An operating system function similar to paging; used to copy virtual memory between physical memory and the paging file (*see* virtual memory). Swapping is almost identical to paging except that swapping is done on a process basis and paging is done on a memory-page basis. Swapping is used when the amount of virtual memory in use has exceeded the amount of physical memory available. Swapping is quite expensive in terms of performance and should be avoided if possible.

synonym. An alias for a table, view, sequence, or program unit.

System Global Area. A shared memory region Oracle uses to store data and control information for one Oracle instance. The SGA is allocated when the Oracle instance starts; it is deallocated when the Oracle instance shuts down. Each Oracle instance that starts has its own SGA. The information in the SGA is made up of the database buffers, the redo log buffer, and the shared pool; each has a fixed size and is created at instance startup.

table. The basic unit of storage in the Oracle database. Users store their data in tables.

tablespace. A logical structure that consists of one or more data files. A tablespace is used to logically contain tables, clusters, and indexes.

TCP/IP (Transmission Control Protocol/Internet Protocol). A network protocol. TCP/IP is probably the most used network protocol in the world.

thread. Sometimes known as a *lightweight process*. Similar to a process but shares the process context with other threads. A thread has much less overhead associated with it than does a normal process. A *thread switch* (change between threads) has much less overhead than a process switch.

Token Ring. A hardware network standard. Token Ring networks use a token-passing mechanism for arbitration. Only the NIC with the token can use the network.

transaction. A set of database statements that represents a logical unit of work or function. A database transaction starts when the first SQL statement is submitted and ends when the commit or rollback has occurred. Performance measurements (such as those described in Chapter 5, “Benchmarking,” often uses the number of transactions per second as the performance metric.

trigger. A mechanism that allows you to write procedures that are automatically executed whenever an `INSERT`, `UPDATE`, or `DELETE` statement is executed on a table or view. Triggers can be used to enforce integrity constraints or automate some other custom function.

two-phase commit. The process by which distributed transactions occur. In a two-phase commit, each node commits its changes and signals that it has completed. When all nodes have successfully committed, the distributed transaction has committed.

UPDATE. The SQL statement used to change rows in a table.

view. A window into a table or set of tables. A view is a way for a table or set of tables to be seen. A view, like a table, can be queried, updated, inserted into, and deleted from. The data, however, is actually stored in the tables to which the view refers.

virtual memory. The memory that can be used for programs in the operating system. To overcome the limitations associated with insufficient physical memory, virtual memory allows programs to run that are larger than the amount of physical memory in the system. When there is not enough physical memory in the system, these programs are copied from RAM to a disk file called a *paging* or *swap file*. This arrangement allows small systems to run many programs. There is a performance penalty you pay when the computer pages or swaps.

Appendix



Oracle Tuning Parameters

This appendix lists the Oracle tuning parameters. The appendix first groups them into general areas of use and then sorts them alphabetically within the group. The section headings are the syntax for the parameters. The syntax contains information in the following format:

PARAMETER[option1, option2, option3, etc. .] <DEFAULT VALUE>

These parameters are divided into sections based on whether the parameter affects performance, is a parameter that enables system analysis, is a general parameter, and so on. Because there may be some overlap, if a parameter is not in the section where you expected it, keep looking.

Performance Parameters

These parameters change the performance characteristics of the system.

CURSOR_SPACE_FOR_TIME [TRUE,FALSE] < FALSE >

`CURSOR_SPACE_FOR_TIME` causes the system to use more space for cursors, thus increasing performance. This parameter affects both the shared SQL areas and the user's private SQL area. This parameter speeds performance but uses more memory.

If `CURSOR_SPACE_FOR_TIME` is TRUE, the shared SQL areas remain pinned in the shared pool as long as an open cursor references them. This parameter should be used only if you have a sufficiently large shared pool to hold all the processes' cursors simultaneously.

The user's private SQL area is also retained during cursor execution, thus saving time and I/Os at the expense of memory.

DB_BLOCK_BUFFERS [4..65535] < 32 buffers >

This parameter controls the number of database block buffers in the SGA. `DB_BLOCK_BUFFERS` is probably the most significant instance tuning parameter because the majority of I/Os in the system are generated by database blocks. Increasing `DB_BLOCK_BUFFERS` increases performance at the expense of memory. You can calculate the amount of memory that will be consumed with the following formula:

`Buffer Size = DB_BLOCK_BUFFERS * DB_BLOCK_SIZE`

A larger number of database block buffers in the system creates a higher cache-hit rate, thus reducing the amount of I/O and CPU utilized and improving performance.

DB_BLOCK_CHECKPOINT_BATCH [0..DB_BLOCK_WRITE_BATCH] < DB_BLOCK_WRITE_BATCH / 4 >

This parameter specifies the number of blocks that the DBWR writes in one batch when performing a checkpoint. Setting this value too high causes the system to flood the I/O devices during the checkpoint, severely degrades performance, and increases response times—maybe

to unacceptable levels.

You should set DB_BLOCK_CHECKPOINT_BATCH to a level that allows a checkpoint to finish before the next checkpoint occurs. Because setting DB_BLOCK_CHECKPOINT_BATCH to zero causes the default value to be used, changing DB_BLOCK_WRITE_BATCH is automatically reflected in the default value of DB_BLOCK_CHECKPOINT_BATCH.

DB_BLOCK_SIZE [1024-8192 (OS dependent)] < OS dependent >

This parameter specifies in bytes the size of the Oracle database blocks. The typical values are 2048 and 4096. If you set the block size relative to the size of the rows in database, you can reduce I/O. In some types of applications in which large amounts of sequential accesses are done, a larger database block size can be of benefit. This value is of use only at database creation time.

DB_FILE_MULTIBLOCK_READ_COUNT [number (OS dependent)] < OS dependent >

DB_FILE_MULTIBLOCK_READ_COUNT specifies the maximum number of blocks read in one I/O during a sequential scan. The default is a function of DB_BLOCK_BUFFERS and PROCESSES. Reasonable values are 4, 16, or 32. The maximum allowed values are OS dependent.

This parameter can be especially useful if you do a large number of table scans such as in a DSS system.

DB_FILE_SIMULTANEOUS_WRITES [1..24] < 4 >

Specifies the number of simultaneous writes for each database file when written by the DBWR. For disk arrays that handle large numbers of requests in the hardware simultaneously, it is advantageous to set DB_FILE_SIMULTANEOUS_WRITES to its maximum.

DISCRETE_TRANSACTIONS_ENABLED [TRUE/FALSE] < FALSE >

This parameter implements a simpler, faster rollback mechanism that, under certain conditions, can improve performance. In this mode, you can obtain greater efficiency but the qualification criteria for what kind of transactions can take advantage of discrete transactions are quite strict.

DML_LOCKS [20..unlimited,0] < 4 * TRANSACTIONS >

This parameter specifies the maximum number of DML locks. A DML lock is used for each table modification transaction. DML locks are used in the `DROP TABLE`, `CREATE INDEX`, and `LOCK TABLE IN EXCLUSIVE MODE` statements. If the value is set to zero, enqueues are disabled, which improves performance slightly.

LOG_ARCHIVE_BUFFER_SIZE [1..OS dependent] < OS dependent >

When running in ARCHIVELOG mode, this parameter specifies the size of each archival buffer in redo log blocks. This parameter, in conjunction with the `LOG_ARCHIVE_BUFFERS` parameter, can be used to tune the speed of archiving faster or slower as desired to affect overall system performance.

LOG_ARCHIVE_BUFFERS [1..OS dependent] < OS dependent >

When running in ARCHIVELOG mode, this parameter specifies the number of buffers to allocate to archiving. This parameter is used with the `LOG_ARCHIVE_BUFFER_SIZE` parameter to control the speed of archiving.

LOG_BUFFER [OS dependent] < OS dependent >

`LOG_BUFFER` specifies the number of bytes allocated to the redo log buffer. Larger values reduce I/Os to the redo log by writing fewer blocks of a larger size. Particularly in a heavily used system, this may help performance.

LOG_CHECKPOINT_INTERVAL [2..unlimited] < OS dependent >

This parameter specifies the number of redo log file blocks to be filled to cause a checkpoint to occur. Remember that a checkpoint always happens when a log switch occurs. This parameter can be used to cause checkpoints to occur more frequently. Sometimes, frequent checkpoints have less effect on the system than one large checkpoint when the log switch occurs.

LOG_CHECKPOINT_TIMEOUT [0..unlimited] < OS dependent >

This parameter specifies the maximum amount of time that can pass before another checkpoint must occur. This parameter can also be used to increase the frequency of the checkpoint process, thus changing the overall system affect.

LOG_SIMULTANEOUS_COPIES [0..unlimited] < CPU_COUNT >

LOG_SIMULTANEOUS_COPIES specifies the number of redo buffer copy latches simultaneously available to write log entries. You can have up to two redo copy latches per CPU. This helps the LGWR process keep up with the extra load generated by multiple CPUs.

If this parameter is zero, redo copy latches are turned off and all log entries are copied on the redo allocation latch.

LOG_SMALL_ENTRY_MAX_SIZE [number (OS dependent)] < OS dependent >

This parameter specifies the size in bytes of the largest copy to the log buffers that can occur under the redo allocation latch without obtaining the redo buffer copy latch. If **LOG_SIMULTANEOUS_COPIES** is zero, this parameter is ignored.

OPTIMIZER_MODE [RULE/COST/FIRST_ROWS/ALL_ROWS] COST

When set to **RULE**, this parameter causes rule-based optimization to be used, unless hints are supplied in the query. When set to **COST**, this parameter causes a cost-based approach for the SQL statement, providing that there are any statistics in the data dictionary. When set to **FIRST_ROWS**, the optimizer chooses execution plans that minimize response time. When set to **ALL_ROWS**, the optimizer chooses execution plans that minimize total execution time.

PRE_PAGE_SGA [TRUE/FALSE] < FALSE >

When set to **TRUE**, this parameter specifies that, at instance startup, all pages of the SGA are touched, causing them to be allocated in memory. This increases startup time but reduces page faults during run time. This is useful if you have a large number of processes starting at once. This parameter can increase the system performance in that case by avoiding memory allocation overhead.

ROLLBACK_SEGMENTS

[Any rollback segment names] < NULL >

ROLLBACK_SEGMENTS specifies one or more rollback segment names to be allocated to this instance. If ROLLBACK_SEGMENTS is not specified, the public rollback segments are used. If you want to move your rollback segments to a different disk device, you must specify it here. The parameter is specified as follows:

```
ROLLBACK_SEGMENTS = ( roll1, roll2, roll3 )
```

If you use the Oracle Parallel Server option, you must name different rollback segments for each instance.

ROW_CACHE_CURSORS [10..3300] < 10 >

This parameter specifies the number of cached recursive cursors used by the row cache manager for selecting rows from the data dictionary. The default is usually sufficient unless you have particularly high access to the data dictionary.

ROW_LOCKING [ALWAYS/INTENT] < ALWAYS >

The value ALWAYS specifies that only row locks are acquired when a table is updated. If you set this value to INTENT, row locks are acquired on a SELECT FOR UPDATE, but when the update occurs, a table lock is acquired.

SEQUENCE_CACHE_ENTRIES [10..32000] < 10 >

This parameter specifies the number of sequences that can be cached in the SGA. By caching the sequences, an immediate response is achieved for sequences. Set a large value for SEQUENCE_CACHE_ENTRIES if you have a high concurrency of processes requesting sequences.

SEQUENCE_CACHE_HASH_BUCKETS

[1..32000 (prime number)] < 7 >

This parameter specifies the number of buckets to speed up access to sequences in the cache. The cache is arranged as a hash table.

SERIALIZABLE [TRUE/FALSE] < FALSE >

If this value is set to TRUE, it causes queries to obtain table-level read locks, which prohibits modifications to that table until the transaction has committed or rolled back the transaction. This mode provides repeatable reads and ensures that, within the transactions, multiple queries to the same data achieve the same result.

With `SERIALIZABLE` set to `TRUE`, degree-three consistency is provided. There is a performance penalty paid when you run in this mode, which is usually not necessary.

SESSION_CACHED_CURSORS [0..OS dependent]

< 0 >

This parameter specifies the number of session cursors to cache. If parse calls of the same SQL statement are repeated, this can cause the session cursor for that statement to be moved into the session cursor cache. Subsequent calls need not reopen the cursor.

SHARED_POOL_SIZE [300KB..OS dependent]

< 3.5Mbytes >

This parameter specifies the size of the shared pool in bytes. The shared pool contains the data dictionary cache (row cache) and the library cache as well as session information. Increasing the size of the shared pool should help performance, at the cost of memory.

SMALL_TABLE_THRESHOLD [0..OS dependent]

< 4 >

This parameter specifies the number of buffers available in the SGA for table scans. A small table may be read entirely into cache if it fits in `SMALL_TABLE_THRESHOLD` number of buffers. When scanning a table larger than this, these buffers are reused immediately. This provides a mechanism to prohibit a single-table scan from taking over the buffer cache.

SORT_AREA_RETAINED_SIZE [0..SORT_AREA_SIZE]

< SORT_AREA_SIZE >

`SORT_AREA_RETAINED_SIZE` defines the maximum amount of session memory in bytes that can be used for an in-memory sort. The memory is released when the last row is fetched from the sort area.

If the sort does not fit in `SORT_AREA_RETAINED_SIZE` bytes, a temporary segment is allocated and the sort is done in this temporary table. This is called an external (disk) sort. This value is important if sort performance is critical.

SORT_AREA_SIZE [number of bytes] < OS dependent >

This value specifies the maximum amount of PGA memory to use for an external sort. This memory is released when the sorted rows are written to disk. Increasing this value increases the performance of large sorts.

Remember that each user process has its own PGA. You can calculate the potential memory usage if all the users are doing a large sort with the following formula:

```
Potential Memory Usage = SORT_AREA_SIZE * ( number of users doing a large sort )
```

If very large indexes are being created, you may want to adjust this parameter up.

SORT_SPACEMAP_SIZE [bytes] < OS dependent >

The size in bytes of the sort spacemap in the context area. If you have very large indexes, adjust this parameter up. Optimal performance is achieved when this parameter has the following value:

```
SORT_SPACEMAP_SIZE = ( total-sort-bytes / sort-area-size ) + 64
```

In this formula, `total-sort-bytes` has the following value:

```
total-sort-bytes = record-count * ( sum-of-average-column-sizes +
( 2 * number-of-columns ) )
```

The `number-of-columns` include the `SELECT` list for the `ORDER BY`, the `GROUP BY`, and the key list for the `CREATE INDEX`. Also add a 10 or 20 extra bytes for overhead.

Parallel Query Option Parameters

The following parameters affect the operation of the Parallel Query option, which has been available in Oracle since version 7.1. The Parallel Query option can affect performance of certain operations dramatically.

PARALLEL_DEFAULT_MAX_SCANS [0..unlimited] < OS dependent >

This value specifies the maximum number of query servers to be used by default for a query. This value is used only if there are no values specified in a `PARALLEL` hint or in the `PARALLEL` definition clause. This limits the number of query servers used by default when the value of `PARALLEL_DEFAULT_SCANSIZE` is used by the query coordinator.

PARALLEL_DEFAULT_SCANSIZE [0..OS dependent] < OS dependent >

This parameter is used to determine the number of query servers to be used for a particular table. The size of the table divided by PARALLEL_DEFAULT_SCANSIZE determines the number of query servers, up to PARALLEL_DEFAULT_MAX_SCANS.

PARALLEL_MAX_SERVERS [0..100] < OS dependent >

This parameter specifies the maximum number of query servers or parallel recovery processes available for this instance.

PARALLEL_MIN_SERVERS [0..PARALLEL_MAX_SERVERS] < 0 >

This parameter determines the minimum number of query servers for an instance. It is also the number of query servers started at instance startup.

PARALLEL_SERVER_IDLE_TIME [0..unlimited] < OS dependent >

This parameter specifies the number of minutes before Oracle terminates an idle query server process.

RECOVERY_PARALLELISM [0..PARALLEL_MAX_SERVERS] < OS dependent >

This parameter specifies the number of processes to be used for instance or media recovery. A large value can greatly reduce instance recovery time. A value of 0 or 1 indicates that parallel recovery will not be done and that recovery will be serial.

Analysis Tool Parameters

The following parameters turn on special features in Oracle for detailed analysis and debugging.

DB_BLOCK_CHECKSUM [TRUE/FALSE] < FALSE >

Setting this parameter to TRUE causes the DBWR and direct loader to calculate a checksum for every block they write to disk. This checksum is written into the header of each block.

DB_LOG_CHECKSUM [TRUE/FALSE] < FALSE >

Setting this parameter to TRUE causes the LGWR to calculate a checksum for every block it writes to disk. The checksum is written into the header of the redo block.

**DB_BLOCK_LRU_EXTENDED_STATISTICS
[0..unlimited] < 0 >**

This parameter enables statistics in the X\$KCBRBH table to be gathered. These statistics estimate the increased number of database block buffer cache hits for each additional buffer. Any value over zero specifies the number of buffers to estimate the cache hits for. If you are interested in estimating the cache hits for an additional 100 buffers, set this parameter to 100.

This parameter affects performance and should be turned off during normal operation.

**DB_BLOCK_LRU_STATISTICS [TRUE/FALSE]
< FALSE >**

This parameter specifies whether statistics are gathered for database block buffer cache-hit estimates as specified in DB_BLOCK_LRU_EXTENDED_STATISTICS. Set this parameter to TRUE when you want to gather these statistics.

EVENT < NULL >

The EVENT parameter modifies the scope of ALTER SESSION SET EVENTS commands so that they pertain to the entire instance rather than just the session. This is an Oracle internal parameter and should be changed only at the direction of Oracle support.

FIXED_DATE [date string] < NULL >

FIXED_DATE allows you to set as a constant the Oracle function SYSDATE in the format YYYY-MM-DD-HH24:MI:SS. Use this parameter for debug only. This parameter allows you to test your application's functionality with certain dates, such as the turn of the century.

SQL_TRACE [TRUE/FALSE] < FALSE >

This parameter specifies whether the SQL Trace facility is enabled. The SQL Trace facility can provide valuable information but at the price of some overhead. Use SQL Trace only when you are tracking down a specific problem.

TIMED_STATISTICS [TRUE/FALSE] < FALSE >

When `TIMED_STATISTICS` is set to TRUE, the time-related statistics in the dynamic performance tables are enabled. This information can be quite useful but there is considerable overhead involved. Only enable `TIMED_STATISTICS` when you are analyzing the system.

General Parameters

These parameters are of a general nature; typically they set limits and do not significantly affect performance—except that they may take up space in the SGA.

BACKGROUND_DUMP_DEST [path-name] < OS dependent >

This parameter specifies the destination directory where the debugging trace files for the background processes are written. The background processes log all startup and shutdown messages and errors to these files, as well as any other error logs. A log of all `CREATE`, `ALTER`, or `DROP` statements is also stored here.

BLANK_TRIMMING [TRUE/FALSE] < FALSE >

If the value of `BLANK_TRIMMING` is TRUE, this allows a data assignment of a string variable to a column value that is smaller (assuming that the truncated characters are blank).

CHECKPOINT_PROCESS [TRUE/FALSE] < FALSE >

This parameter determines whether the CKPT background process is enabled. During a checkpoint, the headers of all the data files must be updated. This task is usually performed by the LGWR process. The job of writing the blocks to disk belongs to the DBWR process. If you notice that the LGWR is slowing down during checkpoints, it may be necessary to enable CKPT to eliminate the extra work that LGWR is doing.

CLEANUP_ROLLBACK_ENTRIES [number] < 20 >

This parameter specifies the number of undo records processed at a time when a rollback occurs. This breaks up the rollback and limits a large rollback from locking out smaller rollbacks.

**CLOSE_CACHED_OPEN_CURSORS [TRUE/FALSE]
< FALSE >**

This parameter specifies whether cursors that have been opened and cached by PL/SQL are automatically closed at commit. A value of FALSE allows these cursors to remain open for further use. If cursors are rarely reused, you can save space in the SGA by setting this value to TRUE. If cursors are reused, you can improve performance by leaving this parameter at the default value of FALSE.

COMPATABLE [variable] < release dependent >

Setting this variable guarantees that the DBMS will remain compatible with the specified release. Some features may have to be limited for the compatibility to be maintained.

**COMPATABLE_NO_RECOVERY [variable]
< release dependent >**

This parameter works like the COMPATABLE parameter except that the earlier version (specified as the parameter) may not work on the current database if recovery is necessary.

**CONTROL_FILES [1..8 filenames]
< OS dependent >**

This parameter specifies the path names of one to eight control files. It is recommended that there always be more than one control file and that they exist on different physical devices.

**DB_DOMAIN
[extension components of a global db name]
< WORLD >**

This parameter specifies the extension components of the global database name consisting of valid identifiers separated by periods (for example, `texas.us.widgets.com`). This allows multiple divisions to each have an ACCOUNTING database that is uniquely identified by the addition of the domain.

DBLINK_ENCRYPT_LOGIN [TRUE/FALSE]

< FALSE >

When you connect to another server, Oracle encrypts the password. If the value of DBLINK_ENCRYPT_LOGIN is FALSE and the connection fails, Oracle tries to connect again with a nonencrypted password. If DBLINK_ENCRYPT_LOGIN is TRUE and the connection fails, Oracle does not attempt a reconnection.

DB_FILES

[min: MAXDATAFILES, max OS dependent]

< OS dependent >

This parameter specifies the maximum number of database files that can be open. This value can be reduced if you want to reclaim space in the SGA. There is no performance degradation by leaving this value high, just additional memory usage in the SGA.

DB_NAME [valid name] < NULL >

This parameter provides a string up to eight characters in length that specifies the name of the database. The following characters are valid:

- ◆ Alphabetic characters
- ◆ Numbers
- ◆ Underscore (_)
- ◆ Pound sign (#)
- ◆ Dollar sign (\$)

No other characters can be used. Double quotation marks (that may be used in the name specified by the user) are removed; they cannot be part of the name. The characters used in the DB_NAME parameter are case insensitive. Thus, **SALES** is the same as **Sales** is the same as **sales**.

ENQUEUE_RESOURCES [10. . 65535] < derived >

This parameter specifies the number of resources that can be locked by the lock manager. The default value is derived from PROCESSES and is usually sufficient. The value is derived from this formula:

```
PROCESSES <= 3; default values = 20
PROCESSES 4-10; default value = ((PROCESSES - 3) * 5) + 20
PROCESSES > 10; default value = ((PROCESSES - 10) * 2) + 55
```

If you use a large number of tables, you may have to increase this value. This value should never exceed DML_LOCKS + DDL_LOCKS + 20 (overhead).

GLOBAL_NAMES [TRUE/FALSE] < FALSE >

This parameter determines whether a database link is required to have the same name as the database to which it connects. Oracle recommends setting this parameter to TRUE to ensure the use of consistent naming conventions for databases and links.

IFILE [parameter file name] < NULL >

This parameter embeds another parameter file into the current parameter file. This can be very useful to separate specific changes from the general changes that you often make. The parameter also allows you to separate different types of parameters such as parallel options.

INIT_SQL_FILES [SQL file name] < NULL >

This parameter lists the names of SQL files that should be run immediately after database creation. This parameter can be used to automatically create the data dictionary.

JOB_QUEUE_INTERVAL [1..3600] < 60 >

This parameter specifies, in seconds, the interval between wake-ups of the SNP background process. The processes run jobs that have been queued.

JOB_QUEUE_KEEP_CONNECTIONS [1..10] < 0 >

This parameter specifies the number of SNP background processes per instance.

JOB_QUEUE_PROCESSES [TRUE/FALSE] < FALSE >

This parameter specifies whether remote connections should be shut down after remote jobs have finished executing.

LICENSE_MAX_SESSIONS

[0..number of session licenses] < 0 >

`LICENSE_MAX_USERS` sets the maximum number of concurrent user sessions allowed. When this limit is reached, only users with `RESTRICTED SESSION` privilege can connect to the server. A zero value indicates that this constraint is not enforced. Either `LICENSE_MAX_USERS` or `LICENSE_MAX_SESSIONS` should be set, not both.

LICENSE_MAX_USERS**[0..number of user licenses] < 0 >**

`LICENSE_MAX_USERS` sets the maximum number of concurrent users you can create in the database. When you reach this limit, you can no longer create users. A zero value indicates that this constraint is not enforced. Either `LICENSE_MAX_USERS` or `LICENSE_MAX_SESSIONS` should be set, not both.

LICENSE_SESSIONS_WARNING**[0..LICENSE_MAX_SESSIONS] < 0 >**

Sets a warning limit so that the administrator can be aware that the `LICENSE_MAX_SESSIONS` limit may soon be reached. After `LICENSE_SESSIONS_WARNING` number of users have connected, a message is written to the alert log for each additional user connecting in.

**LOG_ARCHIVE_DEST [valid path or device name]
< OS dependent >**

When running in ARCHIVELOG mode, this text value specifies the default location and root of the file or tape device to use when archiving redo log files. Archiving to tape is not supported under all operating systems.

**LOG_ARCHIVE_FORMAT [valid filename]
< OS dependent >**

This parameter uses a text string and variables to specify the default filename format of the archive log files. This string is appended to the `LOG_ARCHIVE_DEST` parameter name. The following variables can be used in the string:

%s	Log sequence number
%t	Thread number

Using uppercase letters (%S, %T) causes the value to be fixed length, padded to the left with zeros. A good value is similar to the following:

```
LOG_ARCHIVE_FORMAT = 'log%S_%T.arc'
```

LOG_ARCHIVE_START [TRUE/FALSE] < FALSE >

When running in ARCHIVELOG mode, LOG_ARCHIVE_START specifies whether archiving should be started up automatically at instance startup. A setting of TRUE indicates that archiving is automatic; FALSE indicates that archiving is manual.

**LOG_CHECKPOINTS_TO_ALERT [TRUE/FALSE]
< FALSE >**

This parameter specifies whether you want to log the checkpoints to the alert log. This can be useful in verifying the frequency of checkpoints.

LOG_FILES [2..255] < 255 >

This parameter specifies the maximum number of redo log files that can be opened at instance startup. Reducing this value can save some space in the SGA. If this value is set higher than the value of MAXLOGFILES used at database creation, it does not override MAXLOGFILES.

**MAX_DUMP_FILE_SIZE [0..unlimited] < 500
blocks >**

This parameter specifies the maximum size in OS blocks of any trace file written. Set this if you are worried that trace files may consume too much space.

MAX_ENABLED_ROLES [0..48] < 20 >

This parameter specifies the maximum number of database roles (including subroles) that a user can enable.

MAX_ROLLBACK_SEGMENTS [1..65536] < 30 >

This parameter specifies the maximum number of rollback segments that can be online for one instance.

OPEN_CURSORS [1..OS limit] < 50 >

This parameter specifies the maximum number of open cursors that a single user process can have open at once.

OPEN_LINKS [0..255] < 4 >

This parameter specifies the maximum number of concurrent open connections to remote database processes per user process. This value should exceed the maximum number of remote systems accessed within any single SQL statement.

PROCESSES [6 to OS dependent] < 50 >

This parameter specifies the maximum number of OS user processes that connect to the Oracle instance. This number must take into account the background processes and the login process that started up the instance. Be sure to add an additional 6 processes for the background processes.

REMOTE_LOGIN_PASSWORDFILE [NONE/SHARED/EXCLUSIVE] < NONE >

This parameter specifies whether Oracle checks for a password file. A value of `NONE` indicates that users are authenticated through the operating system. A value of `EXCLUSIVE` indicates that the password file can be used only by one database and can contain names other than `SYS` and `INTERNAL`. Setting this parameter to `SHARED` allows more than one database to use this password file but only `SYS` and `INTERNAL` are recognized by this password file.

SESSIONS [number] < 1.1 * PROCESSES >

This parameter specifies the total number of user and system sessions. Because recursive sessions may occur, this number should be set slightly higher than `PROCESSES`. The `DDL_LOCKS` is derived from this parameter.

SNAPSHOT_REFRESH_INTERVAL [1..3600] < 60 >

This parameter specifies the number of seconds between wake-ups for the instance's snapshot refresh process.

SNAPSHOT_REFRESH_KEEP_CONNECTION [TRUE/FALSE] < FALSE >

This parameter specifies whether the snapshot refresh process should keep remote connections after the refresh. If set to `FALSE`, the remote database connections are closed after the refreshes occur.

SNAPSHOT_REFRESH_PROCESS [0..10] < 0 >

This parameter specifies the number of snapshot refresh processes per instance. You must set this value to 1 or higher for automatic refreshes. One snapshot refresh process is usually sufficient.

SINGLE_PROCESS [TRUE/FALSE] < FALSE >

If **SINGLE_PROCESS** is set to TRUE, the database instance is brought up in a single user mode. A value of FALSE indicates that the database is brought up in a multiprocess mode.

TEMPORARY_TABLE_LOCKS [0..OS dependent] < SESSIONS >

TEMPORARY_TABLE_LOCKS specifies the number of temporary tables that can be created in the temporary segment space. A temporary table lock is required whenever a sort occurs that cannot be held in memory (that is, the sort exceeds **SORT_AREA_RETAINED_SIZE**). If your application contains a large number of **ORDER BY** clauses or you perform a large number of index sorts, you may want to increase this number.

TRANSACTIONS [number] < 1.1 * PROCESSES >

This parameter specifies the maximum number of concurrent transactions in the instance. The default value is greater than **PROCESSES** to provide for recursive transactions. A larger value increases the size of the SGA. If you increase the number of transactions allowed in the system, you may also want to increase the number of rollback segments available.

TRANSACTIONS_PER_ROLLBACK_SEGMENT [1..OS dependent] < 30 >

This value specifies the maximum number of concurrent transactions allowed per rollback segment. You can calculate the minimum number of rollback segments enabled at startup with this formula:

Rollback Segments = TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT

Performance can be improved if there is less contention on rollback segments. In a heavily used system, you may want to reduce **TRANSACTIONS_PER_ROLLBACK_SEGMENT** to decrease this contention.

USER_DUMP_DEST [valid path name] < OS dependent >

USER_DUMP_DEST specifies the path to where the debugging trace files are written.

Multithreaded Server Parameters

These parameters are used if you are using the multithreaded server process.

MTS_DISPATCHERS ["protocol, number "] < NULL >

This parameter specifies the configuration of the dispatcher processes created at startup time. The value of this parameter is a quoted string of two values separated by a comma. The values are the network protocol and the number of dispatchers. Each protocol requires a separate specification. This parameter can be specified multiple times. Here is an example of two dispatcher definitions:

```
MTS_DISPATCHERS = "tcp, 2"
MTS_DISPATCHERS = "ipx, 1"
```

MTS_LISTENER_ADDRESS [configuration] < NULL >

This parameter specifies the configuration of the listener process addresses. There must be a listener process address for each protocol used in the system. Addresses are specified as the SQL*Net description of the connection address.

Because each connection is required to have its own address, this parameter may be specified several times. Here is an example:

```
MTS_LISTENER_ADDRESS = "(ADDRESS=(PROTOCOL=tcp)(HOST=hostname)(PORT=7002))"
MTS_LISTENER_ADDRESS = "(ADDRESS=(PROTOCOL=ipx)())"
```

MTS_MAX_DISPATCHERS [OS dependent] < 5 >

This parameter specifies the maximum number of dispatcher processes allowed to run simultaneously.

MTS_MAX_SERVERS [OS dependent] < 20 >

This parameter specifies the maximum number of shared server processes allowed to run simultaneously.

MTS_SERVERS [OS dependent] < 0 >

This parameter specifies the number of server processes created at instance startup.

MTS_SERVICE [name] < DB_NAME >

This parameter specifies the name of the service to be associated with the dispatcher. Using this name in the CONNECT string allows users to connect using the dispatcher. The name should be unique. Do not specify this name in quotes. Usually, it is a good idea to make this name the same as the instance name. Because the dispatcher is tried first, if it is not available, the CONNECT string can still connect the user to the database through a normal database connection.

Distributed Option Parameters

These parameters are meaningful only when you use the distributed option.

COMMIT_POINT_STRENGTH [0..255] < OS dependent >

This value is used to determine the commit point site when executing a distributed transaction. The site with the highest value for COMMIT_POINT_STRENGTH is the commit point site. The site with the largest amount of critical data should be the commit point site.

DISTRIBUTED_LOCK_TIMEOUT [1..unlimited] < 60 seconds >

DISTRIBUTED_LOCK_TIMEOUT specifies, in seconds, how long distributed transactions should wait for locked resources.

DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME [1..1800] < 200 seconds >

DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME specifies, in seconds, how long to hold a remote connection open after a distributed transaction fails. A larger value holds the connection longer but also continues to use local resources, even though the connection may have been severed. Any value larger than 1800 seconds interferes with the reconnection and recovery background processes and will never drop a failed connection.

DISTRIBUTED_TRANSACTIONS [0..TRANSACTIONS] < OS dependent >

DISTRIBUTED_TRANSACTIONS specifies the maximum number of distributed transactions that the database can process concurrently. This value cannot exceed the value of TRANSACTIONS. If you are having problems with distributed transactions because network failures are causing many in-doubt transactions, you may want to limit the number of distributed transactions.

If DISTRIBUTED_TRANSACTIONS is set to zero, no distributed transactions are allowed and the RECO process does not start at instance startup.

REMOTE_OS_AUTHENT [TRUE/FALSE] < FALSE >

If this parameter is set to TRUE, it allows authentication to remote systems with the value of OS_AUTHENT_PREFIX.

REMOTE_OS_ROLES [TRUE/FALSE] < FALSE >

If this parameter is set to TRUE, it allows remote clients to have their roles managed by the OS. If REMOTE_OS_ROLES is FALSE, roles are managed and identified by the database for the remote system.

Parallel Server Parameters

These parameters are used only in conjunction with the Oracle Parallel Server option.

CACHE_SIZE_THRESHOLD [number] < 0.1 * DB_BLOCK_BUFFERS >

This parameter specifies the maximum size of a cached partition table split among the caches of multiple instances. If the partition is larger than this value, the table is not split among the caches.

GC_DB_LOCKS [0..unlimited] < 0 >

This parameter specifies the number of PCM locks allocated. The value of GC_DB_LOCKS should be at least 1 greater than the sum of the locks specified with the parameter GC_FILES_TO_LOCKS.

GC_FILES_TO_LOCKS**[file_number=locks:filename=locks] < NULL >**

This parameter supplies a list of filenames, each specifying how many locks should be allocated for that file. Optionally, the number of blocks and the value EACH can be added to further specify the allocation of the locks.

GC_LCK_PROCS [0..10] < 1 >

This parameter specifies the number of lock processes (LCK0 to LCK9) to create for the instance. Usually, the default value of 1 is sufficient unless an unusually high number of locks are occurring.

GC_ROLLBACK_LOCKS [number] < 20 >

This parameter specifies the number of distributed locks available for each rollback segment. The default value is usually sufficient.

GC_ROLLBACK_SEGMENTS [number] < 20 >

GC_ROLLBACK_SEGMENTS specifies the maximum number of rollback segments system wide. This includes all instances in the parallel server system, including the SYSTEM rollback segment.

GC_SAVE_ROLLBACK_LOCKS [number] < 20 >

This parameter specifies the number of distributed locks reserved for deferred rollback segments. These deferred rollback segments contain rollback entries for segments taken offline.

GC_SEGMENTS [number] < 10 >

This parameter specifies the maximum number of segments that may have space management activities performed by different instances simultaneously.

GC_TABLESPACES [number] < 5 >

This parameter specifies the maximum number of tablespaces that can be brought online or offline simultaneously.

INSTANCE_NUMBER [1..OS dependent] < lowest available number >

This parameter specifies a unique number that maps the instance to a group of free space lists.

MAX_COMMIT_PROPAGATION_DELAY

This parameter specifies the maximum amount of time that can pass before the SCN (System Change Number) is changed by the DBWR. This value helps in certain conditions where the SCN may not be refreshed often enough because of a high load from multiple instances.

PARALLEL_DEFAULT_MAX_INSTANCES [0..instances] < OS dependent >

This parameter specifies the default number of instances to spit a table among for parallel query processing. This value is used if the `INSTANCES DEFAULT` is specified in the table/cluster definition.

THREAD [0..max threads] < 0 >

This parameter specifies the number of the redo thread to be used by this instance. Any number can be used, but the value must be unique within the cluster.

Security Parameters

The following parameters help set up system security; manipulate them to obtain the best mix of efficiency and security.

AUDIT_TRAIL [NONE,DB,OS]

The `AUDIT_TRAIL` parameter enables auditing to the table `SYSAUD`. Auditing causes a record of database and user activity to be logged. Because auditing causes overhead, it limits performance. The amount of overhead and the effect on performance is determined by what and how much is audited. Once `AUDIT_TRAIL` is enabled, auditing is turned on by the Oracle command `AUDIT`.

OS_AUTHENT_PREFIX [] < OPS\$ >

This is the value concatenated to the beginning of the user's OS login account to give a default Oracle account name. The default value of `OPS$` is OS dependent and is provided for backward compatibility with previous Oracle versions. Typically, you use the default or set the value to `""` (NULL) to eliminate prefixes altogether.

OS_ROLES [TRUE/FALSE] < FALSE >

Setting this parameter to TRUE allows the OS to have control over the username's roles. If set to FALSE, the username's roles are controlled by the database.

SQL92_SECURITY [TRUE/FALSE] < FALSE >

This parameter specifies whether the table-level `SELECT` privileges are needed to execute an update or delete that reference's table column values.

Trusted Oracle7 Parameters

The following parameters apply to the Trusted Oracle7 option.

AUTO_MOUNTING [TRUE/FALSE] < TRUE >

When set to TRUE, this parameter specifies that a secondary database is mounted by the primary database whenever a user connected to the primary database requests data from the secondary database.

DB_MOUNT_MODE

[NORMAL/READ_COMPATIBLE] < NORMAL >

This parameter specifies the access mode to which the database is mounted at instance startup. A value of `NORMAL` starts the database in normal read-write mode; `READ_COMPATIBLE` starts the database in read-write mode with the addition that it supports concurrent mounting by one or more read-secure instances.

LABEL_CACHE_SIZE [number > 50] < 50 >

This parameter specifies the cache size for dynamic comparison of labels. This number should be greater than the label-category combinations in the OS and should never be less than 50.

MLS_LABEL_FORMAT [valid label format] < sen >

This parameter specifies the format used to display labels. The default value `sen` specifies *sensitive*.

OPEN_MOUNTS [0..255] < 5 >

This parameter specifies the maximum number of databases that an instance can simultaneously mount in OS MAC mode. This value should be large enough to handle all the primary and secondary databases you might mount.

National Language Support Parameters

The following parameters are used in the configuration of National Language Support features.

NLS_CURRENCY [character string] < derived from NLS_TERRITORY >

This parameter specifies the string to use as the local currency symbol for the `L` number format element.

NLS_DATE_FORMAT [format mask] < derived from NLS_TERRITORY >

This parameter defines the default date format to use with the `TO_CHAR` and `TO_DATE` functions. The value of this parameter is any valid date format mask. Here is an example:

```
NLS_DATE_FORMAT = 'DD/MM/YYYY'
```

NLS_DATE_LANGUAGE [NLS_LANGUAGE value] < value for NLS_LANGUAGE >

This parameter determines the language to use for the day and month names and date abbreviations (AM, PM, AD, BC).

NLS_ISO_CURRENCY [valid NLS_TERRITORY value] < derived from NLS_TERRITORY >

This parameter defines the string to use as the international currency symbol for the `C` number format element.

NLS_LANGUAGE [NLS_LANGUAGE value]

< OS dependent >

This parameter defines the default language of the database. This specifies the language to use for messages, the language of day and month names, symbols to be used for AD, BC, AM, and PM, and the default sorting mechanisms.

NLS_NUMERIC_CHARACTERS [two characters]

< derived from NLS_TERRITORY >

This parameter defines the characters to be used as the group separator and decimal. The group separator is used to separate the integer groups (that is, hundreds, thousands, millions, and so on). The decimal separator is used to distinguish between the integer and decimal portion of the number. Any two characters can be used but they must be different. The parameter is specified by two characters within single quotes. To set the group separator to , (comma) and the decimal separator to . (period), use the following statement:

```
NLS_NUMERIC_CHARACTERS = ',.'
```

NLS_SORT [BINARY or named linguistic sort]

< derived from NLS_LANGUAGE >

If this parameter is set to BINARY, the collating sequence for ORDER_BY is based on the numeric values of the characters. A linguistic sort decides the order based on the defined linguistic sort. A binary sort is much more efficient and uses much less overhead.

NLS_TERRITORY [territory name]

< OS dependent >

This parameter specifies the name of the territory whose conventions are used for day and week numbering. The parameter also provides defaults for other NLS parameters.

Appendix



Contents of the CD-ROM

This appendix lists the contents of the CD-ROM disc that accompanies this book. Included on the disc are the SQL scripts used to create many of the examples in the book.

SQL Scripts

The following sections give brief descriptions of the SQL scripts used as examples in this book; the scripts are provided on the CD-ROM disc.

Chapter 9

09opt01.sql	Extract library cache-miss percent.
09opt02.sql	Extract library cache-miss rate.
09opt03.sql	Extract data dictionary area cache-miss percent.
09opt04.sql	Extract data dictionary area cache-miss rate.
09opt05.sql	Extract buffer cache information.
09opt06.sql	Extract physical I/O information.
09opt07.sql	Create striped tablespace.
09opt08.sql	Create table on that tablespace with stripe.
09opt09.sql	Create chained rows table and analyze.
09opt11.sql	Check dynamic rollback segment growth.
09opt10.sql	Check recursive calls.
09opt12.sql	Check rollback segment statistics.
09opt13.sql	Check for rollback segment contention.
09opt14.sql	Check for log buffer contention.
09opt15.sql	Check for latch contention.
09opt16.sql	Check sort performance.
09opt17.sql	Check for free list contention.

Chapter 10

10opt01.sql	Look for tablespace fragmentation.
10opt02.sql	Create DOGS table.
10opt03.sql	Create BREEDS table.
10opt04.sql	Load BREEDS table.
10opt05.sql	Load DOGS table.
10opt06.sql	Create dog_space tablespace.
10opt07.sql	Create dog_cluster.
10opt08.sql	Create DOGS2 table in dog_cluster.
10opt09.sql	Create BREEDS2 table in dog_cluster.
10opt10.sql	Create index on dog_cluster.

- 10opt11.sql Load BREEDS2.
 10opt12.sql Load DOGS2.

NOTE: The CD-ROM contains two versions of the same tables: DOGS and DOGS2, and BREEDS and BREEDS2. One set of tables has constraints defined within the table; the other set uses alternative table commands.

Chapter 16

- 16opt01.sql Create a striped tablespace.
 16opt02.sql Create a striped table.

Chapter 25

- 25opt01.sql Create tablespace dog_space and create tables DOGS and BREEDS.
 25opt02.sql Insert rows into BREEDS table.
 25opt03.sql Insert rows into DOGS table.
 25opt04.sql Select sid, serial# and osuser from V\$SESSION.
 25opt05.sql Turn on SQL Trace for session.
 25opt06.sql Turn off SQL Trace for session.
 25opt07.sql Select join from DOGS and BREEDS.
 25opt08.bat Run TKPROF.
 25opt09.sql Create plan_table.
 25opt10.sql EXPLAIN PLAN example.
 25opt11.sql Set up plan_table.
 25opt12.sql Extract execution plan from plan_table.

Chapter 27

- 27opt01.sql ANALYZE TABLE with COMPUTE STATISTICS.
 27opt02.sql ANALYZE TABLE with ESTIMATE STATISTICS.
 27opt03.sql ANALYZE TABLE with ESTIMATE STATISTICS with SAMPLE.
 27opt04.sql ANALYZE TABLE with VALIDATE STRUCTURE.
 27opt05.sql ANALYZE TABLE with LIST CHAINED ROWS INTO.
 27opt06.sql ANALYZE TABLE with LIST CHAINED ROWS.

Chapter 28

28opt01.sql

Create stored procedure old_dogs.

Chapter 29

29opt01.txt	DOGS table in Wordpad format.
29opt02.txt	BREEDS table in Wordpad format.
29opt03.sql	ALTER TABLE BREEDS to ADD PRIMARY KEY(breed).
29opt04.sql	ALTER TABLE DOGS to ADD FOREIGN KEY BREEDS(breed).
29opt05.sql	ALTER TABLE DOGS for CHECK.
29opt06.sql	Add dog that fails constraint.
29opt07.sql	Add dog that passes constraint.
29opt08.sql	CREATE TABLE DOGS and BREEDS with constraints.
29opt09.sql	Select constraint information.
29opt10.sql	Select constraint information for DOGS table.
29opt11.sql	Select column constraint information.
29opt12.sql	Add trigger, old_one.
29opt13.sql	Add an old dog.

Chapter 32

32opt01.sql	CREATE SEQUENCE.
32opt02.sql	Use the sequence.
32opt03.sql	Demonstrate array processing.

Index

A

- ACID tests (Atomicity, Consistency, Isolation, and Durability), 55-58
- ad-hoc, 22, 608
- add-ons, *see* Parallel Query option; Parallel Server option
- AdHawk tool, 498
- administrator (database administrator), 23, 610
- agents, SNMP agents, 76
- aggregate functions, 608
- AIO, *see* Asynchronous I/O
- alerts, triggers, 470
- alias, *see* synonyms
- ALL_CONS_COLUMNS view, 467
- ALL_CONSTRAINTS view, 467
- ALL_ROWS hint, 479
- ALL_TRIGGERS view, 471
- ALTER TABLE command, 466
- analysis tools, 496
 - defined, 490
 - Oracle Mission Control, 496-497
 - Oracle Trace, 497
 - parameters, list of, 627-629
 - summary, 499
 - third-party tools, 497
 - AdHawk*, 498
 - DATA-XPERT*, 498
 - DBGGENERAL*, 498
 - Patrol*, 498-499
 - Precise/SQL suite*, 498
 - TSreorg*, 498

- ANALYZE command, 441-442**
 checking structural integrity, 444
 data dictionary statistics, 445-447
 gathering chained-row statistics, 444-445
 gathering statistics, 442-443
- Analyze/SQL tool, 498**
- AND_EQUAL hint, 481**
- application development tools, 494, 534**
 advantages, 534
 defined, 490
 first-generation tools, 534-535
 modern tools, 535
 Oracle tools, 494, 536
 Developer/2000, 495, 536-544
 Power Objects, 495, 544-546
 third-party tools, 495, 546
 Delphi, 495-496, 547-548
 PowerBuilder, 495, 552
 ReportSmith, 548-549
 SQL Windows, 495, 550-552
 tuning automatically generated SQL statements, 535-536
- application servers**
 defined, 557-558
 tuning, 559
- applications, 17**
 client, bottlenecks, 523
 ConText, 18
 designing, 83, 426
 clusters, 430
 discrete transactions, 435-436
 functions, 432-433
 hash clusters, 431-432
 indexes, 426-429
- optimization techniques, 433-434**
- packages, 432-433**
- procedures, 432-433**
- Media Server, 18**
- Oracle Financial Applications, 17**
- Oracle Office, 17-18**
- Oracle WebServer, 18**
- registering, 417**
- tuning considerations, 420**
 analyzing effects of SQL statements, 424-425
 analyzing SQL statements, 422-424
 problem analysis, 420-421
 summary, 425, 591-592
- ARCH (Archiver Process), 12, 26-27**
- architectures**
 buses, 211-212
 memory, 210
 virtual memory system, 211
 microkernel architecture, 183
 networks
 hardware components, 566-570
 protocols, 570-572
 operating systems
 NetWare, 178-179
 OS/2, 188
 UNIX, 192
 Windows NT, 183-184
- Parallel Server option, 340-343**
- three-tiered system architecture, 557-558**
 tuning, 559
- two-tiered system architecture, 556**
- archive logs**
 batch processing systems, 276
- BLOB systems, 332**
- data warehouses, 314**
- decision support systems, 295**
- defined, 26-27, 134-135**
- financial systems, 372**
- office systems, 385**
- OLTP systems, 255**
- TextServer 3.0 systems, 381**
- WebServer systems, 389**
- ARCHIVELOG mode, backup and recovery process, 351**
- Archiver process (ARCH), 12, 26-27**
- arrays**
 disk arrays, 611
 processing, 510
- Asynchronous I/O (AIO)**
 defined, 22, 171-172, 608
 operating systems, 589
 UNIX, 198-200
- ATM networks, 570**
- atomicity, 55**
- Atomicity, Consistency, Isolation, and Durability (ACID tests), 55-58**
- AUDIT command, 472-473**
- audit trails, 472-473**
- AUDIT_TRAIL parameter, 641**
- AUTO_MOUNTING parameter, 642**
- B**
- B*-Tree index structure, 149**
- background processes**
 ARCH (Archiver), 12
 CKPT (Checkpoint), 11
 fast checkpoints, 26
 normal checkpoints, 26
- DBWR (Database Writer), 11**
 defined, 11-12
- Dnnn (Dispatcher), 12**
- LCK (Parallel Server Lock), 12**
- LGWR (Log Writer), 11**
- PMON (Process Monitor), 11**

- RECO (Recovery process), 12
- SMON (System Monitor), 12
- BACKGROUND_DUMP_DEST parameter, 629**
- backups**
- characteristics, 352
 - cold (offline) backups*, 352
 - data access patterns during backup*, 353
 - goals of backups*, 353-354
 - hot (online) backups*, 352-353
 - loads during backups*, 353
 - cold (offline) backups*
 - defined*, 350, 354
 - hardware considerations*, 355-356
 - physical data access*, 354-355
 - software considerations*, 355-356
 - design considerations, summary, 358
 - enhancements, 359
 - CPU*, 359
 - I/O*, 359-360
 - networks*, 360
 - segmenting backups*, 360-362
 - summary*, 361-362
 - testing database*, 362
 - testing operating system*, 363-365
 - verifying performance*, 362-365
 - exports, 350
 - full backups, 350
 - hot (online) backups
 - defined*, 350, 355-356
 - hardware considerations*, 357-358
 - physical data access*, 356-357
 - software considerations*, 357-358
 - offline backups, 350
 - online tablespace backups, defined, 350
 - processes, 351
 - recovery process*, 351-352
 - software testing, 364-365
 - tuning considerations, 358-359
- bandwidth**
- buses, 212
 - defined, 608
 - network design, 576-577
- batch processing systems**
- archive logs, 276
 - block
 - buffers*, 274
 - size*, 277
 - characteristics, 266-267
 - data access patterns*, 267
 - goals of optimally tuned systems*, 268-269
 - loads*, 267-268
 - summary*, 269
 - checkpoints, 276
 - clusters, 277
 - comparisons to OLTP systems, 265
 - contention, latch contention, 275
 - defined, 27, 608
 - design considerations, 270
 - direct write sorts, 277
 - enhancements, 590
 - benchmarks*, 282-283
 - CPU*, 279
 - hardware*, 279-281
 - high-speed compression*, 280-281
 - high-speed tape*, 280-281
 - I/O*, 279-280
 - list of*, 277
 - networks*, 280
 - Parallel Query option*, 277-278
 - Parallel Server option*, 278
 - summary*, 281
 - testing database*, 281
- testing operating system, 282
- verifying performance, 281-283
- hardware considerations, 274
- hash clusters, 277
- indexes, 277
- library cache, 275
- performance criteria of system, 38
 - checklist*, 39
- physical data layouts, 270-271
 - disk arrays*, 272-273
 - traditional disks*, 271-272
- segments, rollback segments, 275
- tuning considerations Oracle, 274-276
 - server operating systems*, 276-277
- benchmarks**
- batch processing systems, 282-283
 - BLOB systems, 336-337
 - custom benchmarks, 70
 - defined*, 51-52
 - writing benchmarks*, 70-72
 - data warehouses, 320-321
 - decision support systems, 301-302
 - defined, 51-52
 - OLTP systems, 262
 - publications, 69
 - standard benchmarks
 - defined*, 51-53
 - TPC (Transaction Processing Performance Council)*, 53- 57
 - testing, 69
 - UNIX features, 200-203
- Binary Large Objects, see BLOB systems**
- binding variables**, 401
- bits**, 29

- BLANK_TRIMMING**
parameter, 629
- BLOB systems (Binary Large Objects)**
archive logs, 332
block
 buffers, 332
 size, 333
checkpoints, 332
characteristics, 324
 data access patterns, 324
 goals of optimally tuned systems, 325-326
 loads, 324-325
 summary, 326-327
clusters, 333
contention, latch
 contention, 332
defined, 28, 323-324, 608
design considerations, 327
enhancements, 333, 590
 benchmarks, 336-337
 CPU, 334
 hardware, 334-335
 I/O, 334-335
 networks, 335
 Parallel Query option, 333
 summary, 335
 testing database, 336
 testing server operating system, 336
 verifying performance, 335-337
hardware considerations, 331
indexes, 333
library cache, 332
multiblock reads, 332
physical data layout, 328
 disk arrays, 329-331
 traditional disks, 328-329
segments, rollback
 segments, 332
tuning considerations
 Oracle, 332
 server operating systems, 333
- blocks**
buffers
 batch operating systems, 274
 BLOB systems, 332
 cache, 583
 data warehouses, 313
 decision support systems, 294
 defined, 22
 financial systems, 371
 office systems, 384
 OLTP systems, 254
 replicated systems, 376
 TextServer
 3.0 systems, 381
 WebServer systems, 388
defined, 22, 608
indexes, 149
multiblock reads or writes, 152-153
size, 584
 batch processing systems, 277
 BLOB systems, 333
 changing, 140-141
 data warehouses, 315
 decision support systems, 296
 financial systems, 372
 OLTP systems, 256
 testing disk block size in backup process, 363
 testing tape drives for backup process, 363
 TextServer 3.0 systems, 381
 WebServer systems, 389
- body**
functions, 454
packages, 458
procedures, 453
- bottlenecks**
clients, performance, 522
 application, 523
 hardware, 524
- network, 523**
presentation, 524
- defined, 92
finding, 92
removing, 93
- branch blocks, indexes, 149**
- bridges, networks, 577**
- buffer**
- buffers**
block buffers, 608
 batch operating systems, 274
 BLOB systems, 332
 cache, 583
 data warehouses, 313
 decision support systems, 294
 defined, 22
 financial systems, 371
 office systems, 384
 OLTP systems, 254
 replicated systems, 376
 TextServer 3.0 systems, 381
 WebServer systems, 388
cache, 9, 22, 608
 cache-hits ratio, 107-108
 SQL statement, 597
 tuning, 107-108
 UNIX operating systems, 193
clean buffers, 22, 608
defined, 22, 608
dirty buffers, 22, 608
disk drive, 215
redo log buffer, 10, 608
 contention, 600-601
 latch contention, 131-132
- buses**
architecture, 211-212
bandwidth, 212, 608
defined, 211
- business models, 27-28**
 see also financial systems
- bytes, 29**

C**cache**

buffer cache, 9, 22, 583
cache-hits ratio, 107-108
SQL statement, 597
tuning, 107-108
UNIX operating systems, 193
CPU cache, 206-207, 210
data dictionary cache, 582
SQL statement, 596-597
defined, 22, 608
disk arrays
controller caches, 228
read/write caches, 229
write caches, 228-229
hits, 609
increasing in library cache, 100-102
ratio in buffer cache, 107-108
library cache, 450-452, 582
batch processing systems, 275
BLOB systems, 332
decision support systems, 295
financial systems, 371
hits, 451
misses, 451
office systems, 384
replicated systems, 376
SQL statement, 596
stored functions and procedures, 456-457
TextServer 3.0 systems, 381
WebServer systems, 388

cache affinity, 174

batch processing systems, 276
BLOB systems, 333
data warehouses, 314
decision support systems, 296
OLTP systems, 255
operating systems, 589

UNIX

disabling preemptive scheduling, 202
summary of tuning guidelines, 202-203

CACHE hint, 485**cache miss, shared pool**, 101**CACHE_SIZE_THRESHOLD parameter**, 347, 639**calls, recursive calls**, 118**Cartesian products**, 505-506, 609**CD-ROM, SQL scripts on**, 646-648**Central Processing Unit**, *see CPU***chaining rows**, 117-118

gathering chained-row statistics, 444-445
SQL statement, 599

CHAR data type, 510**CHECK constraint**, 465**CHECKPOINT_PROCESS parameter**, 629**checkpoints**, 11, 26, 133-134

batch processing systems, 276
BLOB systems, 332
data warehouses, 314
decision support systems, 295
defined, 26
fast checkpoints, 26, 134
financial systems, 372
normal checkpoints, 26, 133

office systems, 385

OLTP systems, 255

TextServer 3.0 systems, 381

WebServer systems, 389

checksums, 23, 609**CHOOSE hint**, 479**CISC processors (Complex Instruction Set Computer)**, 207-208**CKPT**, *see checkpoints***classes, Oracle Power Objects**, 545-546**classroom training**, 19**CLEANUP_ROLLBACK_ENTRIES parameter**, 630**client processes**, 10**client/server systems**

defined, 519-520

performance criteria, 32-33

checklist, 35-36

client component, 33

network, 35

server component, 33-35

transactions, running, 32

clients

bottlenecks, performance, 522

application, 523

hardware, 524

network, 523

presentation, 524

computing models, 520

three-tiered system, 521-522

two-tiered system, 520-521

defined, 32, 516

development of

client/server model, 519-520

GUI/Server model, 517-519

network computing model, 517

traditional computing model, 516-517

performance criteria, 33

tuning considerations, 85

summary, 592

CLOSE_CACHED_OPEN_CURSORS parameter, 630**CLUSTER hint**, 481**cluster key**, 609**clusters**, 584

advantages, 141-143

batch processing systems, 277

- BLOB systems, 333
 computers, 609
 creating, 430
 data warehouses, 315
 decision support
 systems, 296
 defined, 6
 design stage, 83
 disadvantages, 142-143
 financial systems, 372
 index clusters, 141, 427,
 609
 OLTP systems, 256
 Parallel Server option, 340
 tables, 609
 TextServer 3.0 systems, 381
 views, data in, 446
 WebServer systems, 389
see also hash clusters
- cold (offline) backups**
 defined, 350, 352-354
 hardware/software
 considerations, 355-356
 physical data access,
 354-355
- cold data**, 609
- cold databases**, 609
- collisions**, 609
- columns**
 indexes, choosing columns
 to index, 428
 indexing, data selection
 guidelines, 151
 tables, 6
 views, data in, 447
- commands**
 ALTER TABLE, 466
 ANALYZE, 441-442
 *checking structural
 integrity*, 444
 data dictionary statistics,
 445-447
 *gathering chained-row
 statistics*, 444-445
 gathering statistics,
 442-443
- AUDIT, 472-473
 CREATE SEQUENCE,
 parameters, 502-503
 CREATE TABLE, 466
 CREATE TRIGGER, 469
 DDL (Data Definition
 Language), 23, 610
 DML (Data Manipulation
 Language), 23, 611
 EXPLAIN PLAN
 *analyzing SQL statement
 execution*, 424
 defined, 394-395,
 404, 414
 extracting results of, 416
 initialization, 414-415
 invoking, 415-416
 NOAUDIT, 472-473
- COMMIT_POINT_STRENGTH**
 parameter, 638
- COMPATABLE**
 parameter, 630
- COMPATABLE_NO_**
RECOVERY parameter,
 630
- Complex Instruction Set**
**Computer (CISC
 processors)**, 207-208
- complex statements**, 609
- component queries**, 609
- composite indexes**, 149
 advantages, 151
 creating, 428-429
 defined, 427
- compound queries**, 609
- compression, high-
 performance com-
 pression**, 365
- computing**
 client/server models,
 519-520
 three-tiered system,
 521-522
 two-tiered system,
 520-521
 development of, 516-520
 GUI/Server model,
 517-519
- network computing
 model*, 517
*traditional computing
 model*, 516-517
- concurrency**, 23, 610
- configuring disk array**
 systems, 240
- connectivity**, 43
- consistency**, 55
- consistent mode**, 610
- consistent read**, 610
- constraints**, 466
 CHECK, 465
 defined, 610
 FOREIGN, 465
 implementing
 at table creation, 466
 at table modification,
 466-467
 integrity constraints, list
 of, 465
 NOT NULL, 465
 PRIMARY, 465
 summary, 469
 UNIQUE, 465
 viewing, 467-469
- consulting services**, 18
- contention**
 defined, 23, 610
 disk contention, 109-110
 identifying problems,
 110-111
 isolating sequential I/Os,
 112
 *separating data from
 index*, 116
 striping random I/Os,
 112-116
 free list contention,
 136-137
 SQL statement, 602
- latch contention,
 130-131, 314
 *batch processing
 systems*, 275
- BLOB systems, 332

- data warehouses*, 314
decision support systems, 295
financial systems, 371
office systems, 385
OLTP systems, 255
redo log buffer, 131-132
replicated systems, 376
SQL statement, 601
TextServer 3.0 systems, 381
WebServer systems, 389
redo log buffer contention, 130-131
rollback contention, 123
 financial systems, 371
 office systems, 385
 OLTP systems, 254
 replicated systems, 376
 SQL statement, 599-600
 WebServer systems, 389
 SQL statement, 600-601
- ConText**, 18
- control files**, 5
- CONTROL_FILES parameter**, 630
- controller**, 227
- caches, 228
 - fiber optic networks, 569
 - read/write caches, 229
 - write caches, 228
- conversions**, 28
- binary storage units, 29-30
 - powers of 10, 29
- copying databases**, *see replicated systems*
- core**
- OS/2, 188
 - Windows NT, 183
- cost-based optimization method**
- ANALYZE command*, 441-442
 - checking structural integrity*, 444
 - data dictionary statistics*, 445-447
- gathering chained-row statistics*, 444-445
gathering statistics, 442-443
choosing, 438-439
defined, 434, 441, 610
hints, 447
- count parameter, SQL Trace**, 409
- CPU (Central Processing Unit)**, 206
- cache, 206-207, 210
 - CISC processors (Complex Instruction Set Computer), 207-208
 - defined, 206-207
 - enhancements
 - backup process*, 359
 - batch processing system*, 279
 - BLOB system*, 334
 - data warehouse*, 317
 - decision support system*, 298-299
 - OLTP*, 257-258
 - replicated system*, 377
 - multiprocessor systems, 209
 - MPP (*Massively Parallel Processor*), 209-210
 - SMP (*Symmetric Multiprocessor*), 209
- RISC processors (Reduced Instruction Set Computer), 208
- cpu parameter, SQL Trace**, 409
- CREATE SEQUENCE command, parameters**, 502-503
- CREATE TABLE command**, 466
- CREATE TRIGGER command**, 469
- current mode**, 610
- current parameter, SQL Trace**, 409
- current read**, 610
- cursor**, 23, 610
- CURSOR SPACE FOR TIME parameter**, 254
- custom benchmarks**
- defined, 51-52
 - writing benchmarks, 70-71
 - analysis*, 72
 - design*, 71
 - implementation*, 71-72
- Customer-Demographics transactions**, OLTP, 65
- Customer-Inquiry transactions**, OLTP, 65
- Customer-Status transactions**, OLTP, 65
- cylinders (platters)**, 215
- D**
- data access patterns**
- backup process*, 353
 - batch processing systems*, 267
 - BLOB systems*, 324
 - consistent read*, 610
 - data warehouses*, 305-306
 - decision support systems*, 287-288
 - OLTP systems*, 246-247
- data blocks**, 8
- see also* extents; segments
- Data Definition Language**, *see DDL*
- data dictionary**
- cache, 103-104, 582
 - defined, 5, 23, 610
- data files**
- defined, 5
 - logical layer, 5-6
- data generation, triggers**, 469
- data integrity**, *see integrity*
- Data Manipulation Language**, *see DML*
- data segments**, 8
- data transfer rate**, 217-218
- data validation, triggers**, 469

data warehouses
archive logs, 314
block
 buffers, 313
 size, 315
characteristics, 304-305
 data access patterns,
 305-308
 loads, 306-307
 summary, 307-308
checkpoints, 314
clusters, 315
contention, latch
 contention, 314
defined, 28, 610
design considerations, 308
 fault tolerance, 311
 hardware, 312
 physical data layout,
 308-311
direct-write sorts, 315
enhancements, 590
 benchmarks, 320-321
 CPU, 317
 hardware, 317-319
 I/O, 317-318
 list of, 315
 networks, 319
 Parallel Query option, 316
 Parallel Server option,
 316-317
 summary, 319
 testing database, 320
 testing operating system,
 320
 verifying performance,
 319-321
fault tolerance
 considerations, 311
hardware
 considerations, 312
hash clusters, 315
indexes, 315
library cache, 313
multiblock reads, 313
physical data layout
 disk arrays, 310-311
 traditional disks,
 309-310
segments, rollback
 segments, 313
tuning considerations
 Oracle, 312-314
 server operating systems,
 314-315
DATA-XPERT tool, 498
database administrator (DBA), 610
database design tools
 defined, 490
 Designer/2000, 490-492
 generators, 492
 Process Modeller, 491
 Systems Designer,
 491-492
 Systems Modeller, 491
 third-party tools, 492-493
 ERwin/ERX, 493
 S-Designor, 493
 SQL Coder, 493-494
database replication, *see also replicated systems*
Database Trigger Editor (Procedure Builder tool), 542
Database Writer (DBWR), 11
databases
 blocks, 608
 cold databases, 609
 design stage, layout
 considerations, 82
 hot databases, 611
 logical layer
 data blocks, 8
 data dictionary, 5
 data files, 5-6
 defined, 5
 extents, 8
 schemas, 25
 segments, 7
 tablespaces, 5-6
 offline databases, 613
 online databases, 613
 physical layer
 control files, 5
 data files, 5
 defined, 4
 redo log files, 5
schema
 defined, 6-7
 objects, 6
data dictionary cache, SQL statement, 596-597
DB_BLOCK_BUFFERS parameter, 620
DB_BLOCK_CHECKPOINT_BATCH parameter, 620-621
DB_BLOCK_CHECKSUM parameter, 628
DB_BLOCK_LRU_EXTENDED_STATISTICS parameter, 628
DB_BLOCK_LRU_STATISTICS parameter, 628
DB_DOMAIN parameter, 630
DB_FILE_MULTIBLOCK_READ_COUNT parameter, 621
DB_FILE_SIMULTANEOUS_WRITES parameter, 621
DB_FILES parameter, 631
DB_LOG_CHECKSUM parameter, 628
DB_MOUNT_MODE parameter, 642
DB_NAME parameter, 631
DBA (Database Administrator), 23, 610
DBA_CONS_COLUMNS view, 467
DBA_CONSTRAINTS view, 467
DBA_TRIGGERS view, 471
DBGGENERAL tool, 498
DBLINK_ENCRYPT_LOGIN parameter, 631
DBWR (Database Writer), 11
DDL (Data Definition Language) commands, 23, 610
deadlocks, 510, 611
decision support systems
 archive logs, 295
 block size, 296

-
- characteristics, 287
data access patterns,
 287-288
goals of optimally tuned systems, 289
loads, 288-289
summary, 289-290
- checkpoint, 295
- clusters, 296
- contention, latch
 contention, 295
- defined, 28, 611
- design considerations, 290
physical data layout,
 291-293
- direct write sorts, 296
- enhancements,
 296-297, 590
benchmarks, 301-302
CPU, 298-299
hardware, 298-300
I/O, 299-300
networks, 300
Parallel Query option, 297
Parallel Server option, 297-298
summary, 300
testing database, 301
testing operating system, 301
verifying performance,
 300-302
- hardware
 considerations, 294
- hash clusters, 296
- indexes, 296
- library cache, 295
- multiblock reads, 295
- physical data layout, 291
disk arrays, 292-293
traditional disks,
 291-292
- segments, rollback
 segments, 295
- tuning considerations
Oracle, 294-295
server operating system,
 295-296
- declarations**
 functions, 454
 packages, 458
 procedures, 453
- deferred frames, 611**
- DELETE statement, 611**
 displaying execution plan
 with EXPLAIN PLAN
 command, 414-416
- Delivery transaction (TPC-C benchmark), 62**
- Delphi, 495-496, 547**
 Delphi Forms Developer,
 547-548
- design considerations, 82**
 applications, 83, 426
clusters, 430
discrete transactions,
 435-436
functions, 432-433
hash clusters, 431-432
indexes, 426-429
optimization techniques,
 433-434
packages, 432-433
procedures, 432-433
- backup process, 354
cold (offline) backups,
 354-355
hot (online) backups,
 355-358
summary, 358
- batch processing systems
hardware, 274
physical data layout,
 270-273
- BLOB systems, 327
hardware, 331
physical data layout,
 328-331
- clusters, 83
- data warehouses, 308
- database layout, 82
- decision support
 systems, 290
hardware, 294
- distributed systems,
 378-379
- financial systems, 369-371
- hardware selection, 83-84
- indexes, 83
- networks, 84, 576
bandwidth, 576-577
bridges, 577
hubs, 577
routers, 577
segmenting networks, 577
- office systems, 384
- Parallel Server option, 343
goals, 343-345
- physical data layout,
 308-309
- replicated systems, 375-377
goals of optimally tuned systems, 375
- systems, list of, 590-591
- TextServer 3.0 systems,
 380-381
- WebServer systems,
 387-388
goals of optimally tuned systems, 387
- Designer/2000, 16, 82, 490-492**
 generators, 492
 Process Modeller, 491
 Systems Designer, 491-492
 Systems Modeller, 491
- Developer/2000, 16, 82, 495**
 features, 536-544
 Forms Designer tool,
 536-538
 Graphics Designer tool,
 537, 540-542
 Procedure Builder tool,
 537, 542-544
 Reports Designer tool, 536,
 538-540
- development tools, see application development tools**
- device drivers, 611**
- Dircect FS interface, 171**
- direct I/O, see synchronous I/O**

- Direct Write Sort option (Parallel Query option), 158**
- direct write sorts, 143, 158**
- batch processing systems, 277
 - data warehouses, 315
 - decision support systems, 296
- disabling**
- SQL Trace
 - for an instance, 406
 - per-session basis, 405-406
 - SQL Trace command, 410
- discrete transactions, 435-436**
- DISCRETE_TRANSACTIONS_ENABLED parameter, 621**
- disk arrays**
- advantages, 226
 - batch processing systems, 272-273
 - BLOB systems, 329-331
 - configuring RAID, 235-236
 - distributing random I/Os, 237-238
 - isolating sequential I/Os, 236-237
 - sizing logical volume, 238-239
 - configuring systems, 240
 - data warehouses, 310-311
 - decision support systems, 292-293
 - defined, 24, 225-226, 611
 - fault-tolerance concerns, 233
 - Full Data Protection mode, 233-234*
 - No Data Protection mode, 233*
 - Partial Data Protection mode, 233-235*
 - summary, 235*
 - hardware arrays
 - controller, 227-228
 - controller caches, 228
 - read/write caches, 229
 - write caches, 228-229
 - hot-swappable disk drives, 228
 - I/O rates, 234
 - logical disks, defined, 24
 - OLTP systems, 252-253
 - operations, 226
 - RAID, 229
 - comparing RAID levels, 240-241
 - RAID-0, 230
 - RAID-1, 230-231
 - RAID-2, 231
 - RAID-3, 231
 - RAID-4, 232
 - RAID-5, 232
 - summarizing RAID levels, 232
 - software arrays, 227
- disk bound systems, 108**
- disk I/O, 108-109**
- contention, 109-110
 - checkpoints, 133-134
 - free list contention, 136-137
 - identifying problems, 110-111
 - isolating sequential I/Os, 112
 - latch contention, 130-132
 - rollback segments, 123-130
 - separating data from indexes, 116
 - striping random I/Os, 112-116
 - data transfer rate, 217-218
 - disk operation, 214-215
 - interfaces, list of, 171
 - queue time, 218
 - random I/Os, 220
 - rotational latency, 217
 - averages, 219
 - seek time, 216-217
 - averages, 219
- sequential I/Os, 220-222
- SQL statement, 597-598
- disk parameter, SQL Trace, 409**
- disks**
- block size, testing in backup process, 363
 - defined, 214
 - drive, 214
 - buffer, 215
 - cylinder, 215
 - data transfer rate, 217-218
 - hot-swappable disk drives, 228
 - platters, 214
 - queue time, 218
 - rotational latency, 217
 - tracks, 214
 - fragmentation, 144-146
 - hot spots, 108
 - logical disks, 24, 612
 - storage capacity, 98
- Dispatcher processes (Dnnn process), 12**
- Distributed Lock Manager, see DLM**
- distributed option**
- defined, 14
 - parameters, list of, 638-639
- distributed systems**
- characteristics, 378
 - defined, 368, 377-378
 - design considerations, 378-379
 - summary of, 379
 - transactions, 378
 - tuning considerations, 378-379
- DISTRIBUTED_LOCK_TIMEOUT parameter, 638**
- DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME parameter, 638**
- DISTRIBUTED_TRANSACTIONS parameter, 639**

DLM (Distributed Lock Manager), 162

Parallel Server option,
341-342

DML (Data Manipulation Language) commands, 23, 611**DML_LOCKS parameter, 622****Dnnn process (Dispatcher processes), 12****drivers, device drivers, 611****drives, disk drives**

buffer, 215
cylinder, 215
data transfer rate, 217-218
defined, 214
hot-swappable disk
drives, 228
platters, 214
queue time, 218
rotational latency, 217
testing block size for
 backup process, 363
tracks, 214

DSS, *see* decision support systems**durability, 56****dynamic performance tables**

buffer cache statistics, 107
defined, 24, 611
V\$FILESTAT, disk access information, 597
V\$LATCH, latch
 contention data, 601
V\$ROLLSTAT, dynamic rollback growth data, 600
V\$ROLLSTAT table,
 rollback segment data, 129
V\$SYSSTAT, recursive calls data, 599
V\$WAITSTAT
 rollback contention data, 599
 rollback segment data, 127
views, table of, 74-75

E**educational services, 19****elapsed parameter, SQL Trace, 409****elapsed time, SQL Trace, 412****enabling**

SQL Trace
 for an instance, 406
 per-session basis, 405

SQL Trace command, 409

enhancements, 583

block size, 584
clusters, 584
fragmentation reductions, 584-585
hash clusters, 585
indexes, 585
multiblock reads and writes, 586
Parallel Query option, 586-587
Parallel Server option, 587-588
spin counts, 588

ENQUEUE_RESOURCES parameter, 631**equijoins, 505, 611****errors, hints, 477-478****ERwin/ERX tool, 493****Ethernet, 566-568, 611**

packets, 567

Etherplex board, OLTP enhancements, 259**EVENT parameter, 628****events, logging triggers, 469****exception handlers, 455****EXCEPTION statement, 455****Exclusive Lock mode,**

Parallel Server option, 342

executing SQL

statements, 401

execution plans, displaying, 414-416**EXPLAIN PLAN command**

analyzing SQL statement execution, 424

defined, 77, 394-395,

404, 414

extracting results of, 416

initialization, 414-415

invoking, 415-416

explicit locking, 509**exports, 350****extents, 8, 611**

see also data blocks;
 segments

F**fast checkpoints, 26, 134****fault-tolerance**

comparisons for disk array

RAID levels, 240-241

data warehouses, 311

disk arrays, 233

Full Data Protection mode, 233-234

No Data Protection mode, 233

Partial Data Protection mode, 233-235

summary, 235

goals, 43

Parallel Server option, 340

FDR (Full Disclosure Report), TPC benchmarks, 56-57**fiber optic networks (FDDI networks), 569-570****Fibre Channel networks, 570****file systems, UNIX, 198****files**

logical layer, data files, 5-6

physical layer, 5

redo log files, 614

financial systems, 17

archive logs, 372

block

buffers, 371

size, 372

characteristics, 369

checkpoints, 372

clusters, 372

- contention
 latch contention, 371
 rollback contention, 371
defined, 368
design considerations, 369-371
 goals for optimally tuned systems, 369
enhancements, 372-373
hash clusters, 372
indexes, 372
library cache, 371
reads and writes, 372
Parallel Query option, 372
segments, rollback
 segments, 371
spin counts, 371
summary of, 373
tuning considerations, 371-372
- FIRST_ROWS hint**, 479-480
- FIXED_DATE**
 parameter, 628
- flowcharts**
 Oracle optimizer, 605
problem-solving
 methodology, 604
SQL statement
 processing, 605
user-transaction
 profile, 605
- FOREIGN constraint**, 465
- foreign keys**, 611
- Forms Designer tool**
 Designer/2000, 536-538
 Oracle Power Objects, 544-545
- Forms Developer (Delphi)**, 547-548
- Forms Generator tool**, 492
- formulas, checksums**, 23, 609
- fragmentation**, 144-146
 reducing, 584-585
- frames**
 deferred frames, 611
 defined, 613
- free list contention**, 136-137
 SQL statement, 602
- front-end computer**, *see clients*
- full backups**, 350
- Full Data Protection mode (fault-tolerance)**, 234
- Full Disclosure Report (FDR)**, 67-69
 TPC benchmarks, 56-57
- FULL hint**, 481
- functions**, 432-433
 aggregate functions, 608
 body, 454
 comparisons to, 452
 declarations, 454
 defined, 24, 452-454, 611
 EXCEPTION, 455
 packages, 24
 parameters, 454
 PL/SQL language, 454-455
 RDBMS_OUTPUT
 package, 456
 RETURN, 454-455
 return values, 454
 stored functions
 creating, 456-457
 defined, 452
 properties, 452
 replacing, 457
 syntax, 453
 see also procedures
- G**
- GB (gigabyte)**, 30
- GC_DB_LOCKS parameter**, 347, 639
- GC_FILES_TO_LOCKS parameter**, 347, 640
- GC_LCK_PROCS parameter**, 347, 640
- GC_ROLLBACK_LOCKS parameter**, 347, 640
- GC_ROLLBACK_SEGMENTS parameter**, 640
- GC_SAVE_ROLLBACK_LOCKS parameter**, 640
- GC_SEGMENTS parameter**, 640
- GC_TABLESPACES parameter**, 640
- generating data, triggers, 469
- generators, 492
- gigabyte (GB)**, 30
- GLOBAL_NAMES parameter**, 632
- goals (tuning)**, 42
 backup process, 353-354
 connectivity, 43
 design goals
 financial systems, 369
 Parallel Server option, 343-345
 replicated systems, 375
 WebServer systems, 387
 fault tolerance, 43
 load time, 44
 response time, 42-43
 setting goals for optimal performance, 48-50
 throughput, 42
 tuning considerations
 batch processing systems, 268-269
 BLOB systems, 325-326
 data warehouses, 307-308
 decision support systems, 289
 OLTP systems, 248-249
 operating systems, 589
- Graphics Designer tool (Designer/2000)**, 537, 540-542
- H**
- HAL (Hardware Abstraction Layer)**, 184
- hardware**
 backup process
 cold (offline) backups, 355

- hot (online) backups*, 357-358
 batch processing systems, 274
enhancements, 279-281
 BLOB systems, 331
enhancements, 334
 bottlenecks
client, 524
finding, 93
 buying considerations, 83-84
 CPU (Central Processing Unit), 206-207
 data warehouses, 312
enhancements, 317-319
 decision support systems, 294
enhancements, 298-300
 network components, 566
Ethernet networks, 566-568
fiber optic networks, 569-570
new technologies, 570
Token Ring networks, 568-569
 OLTP systems, 253, 257
CPU enhancements, 257-258
Etherplex board, 259
I/O enhancements, 258-259
network enhancements, 259
 striping, 114-115
 tuning, 531-532
resources, 94
- Hardware Abstraction Layer (HAL)**, 184
- hardware disk arrays**,
controller, 227-228
 caches, 228
 read/write caches, 229
 write caches, 228-229
- hash clusters**, 146-147, 585
 advantages, 147-148
 batch processing systems, 277
- creating, 431-432
 data warehouses, 315
 decision support systems, 296
 disadvantages, 147-148
 financial systems, 372
 OLTP systems, 256
 TextServer 3.0 systems, 381
 WebServer systems, 389
see also clusters
- HASH hint**, 482
- hierarchy**
 databases
logical layer, 5-8
physical layer, 4-5
- instance, 8
memory structure, 8-10
processes, 10-13
transactions, 12-13
- high-performance compression, backup process**, 365
- high-speed compression, batch processing systems**, 280-281
- high-speed tape, batch processing systems**, 280-281
- hints**
 access methods, 481
AND_EQUAL hint, 481
CLUSTER hint, 481
FULL hint, 481
HASH hint, 482
INDEX hint, 482
INDEX_ASC hint, 483
INDEX_DESC hint, 483
ORDERED hint, 484
ROWID hint, 483
USE_CONCAT hint, 483-484
USE_MERGE hint, 484
USE_NL hint, 485
- advantages, 476
 errors, 477-478
 examples, 477-478
 implementing, 476-478
 multiple, 478
- optimization techniques, 447-478
ALL_ROWS hint, 479
CHOOSE hint, 479
FIRST_ROWS hint, 479-480
RULE hint, 480-481
- Parallel Query option, 485
CACHE hint, 485
NOCACHE hint, 486
NOPARALLEL hint, 486
PARALLEL hint, 485-486
PUSH_SUBQ hint, 486
- syntax, 477
- host-based systems**, *see terminal-based systems*
- hot (online) backups**, 350-353, 355-356
 hardware considerations, 357-358
 physical data access, 356
isolating networks, 357
isolating tablespaces, 356
temporary backup space, 357
 software considerations, 357-358
- hot data**, 611
- hot databases**, 611
- hot spots**, 108
- hot-swappable disk drives**, 228
- HPPI (High Performance Parallel Interface) networks**, 570
- hubs, network design considerations**, 577
- I**
- I/O (Input/Output)**
 asynchronous I/O, 171-172
 backup process, enhancements, 359-360
 batch processing systems, 276
enhancements, 279-280

- BLOB systems, 333
 enhancements, 334-335
- checkpoints, 133-134
- contention, 109-110
 free list contention,
 136-137
 identifying contention problems, 110-111
 latch contention,
 130-132
- data transfer rate, 217-218
- data warehouses, 314
 enhancements, 317-318
- decision support systems, 296
 enhancements, 299-300
- defined, 108-109, 170-171, 612
- Direct FS interface, 171
- disk arrays, 234
- interfaces, list of, 171
- OLTP systems, 255
 enhancements, 258-259
- operating systems, 589
 NetWare, 182-183
 OS/2, 190-191
 UNIX, 197-200
 Windows NT,
 186-187, 528
- operation, 214-215
- queue time, 218
- random I/O
 defined, 24, 220, 598, 614
 disk rates, 598
 distributing in disk arrays, 237-238
 spreading out, 590, 598
 striping data, 112-116
- raw devices, 171
- reducing overhead, 117
 chaining rows, 117-118
 checking for recursive calls, 118-119
 dynamic extensions, 118-119
 migrating rows, 117-118
- PCTFREE command option*, 119-122
- PCTUSED command option*, 119-122
 techniques, 122
- replicated system enhancement, 377
- rotational latency, 217-219
- seek time, 216-217, 219
- separating data from indexes, 116
- sequential I/O
 defined, 25, 111,
 220-222, 598, 615
 disk rates, 598
 isolating, 112, 590, 598
 isolating for disk arrays,
 236-237
- SQL statement, 597-598
- synchronous I/O, 171-172
- tuning considerations, 590
- I/O bound systems, 108**
- IFILE parameter, 632**
- INDEX hint, 482**
- index segments, 8**
- INDEX_ASC hint, 483**
- INDEX_DESC hint, 483**
- indexes, 585**
 avoiding, 429
 B-Tree structure*, 149
 batch processing systems, 277
 BLOB systems, 333
 blocks, 149
 choosing
 columns to index, 428
 tables to index, 427-428
- cluster indexes
 advantages, 141-165
 creating, 430
 defined, 141, 427, 584
 disadvantages, 142-143
 see also hash clusters
- composite indexes
 advantages, 151
 creating, 428-429
 defined, 149, 427
- creating, 149-150, 426-429
 parallel index creation, 159
- data selection guidelines, 150-152
- data warehouses, 315
- decision support systems, 296
- defined, 6, 148-152, 612
- design stage, 83
- financial systems, 372
- nonunique indexes, defined, 149, 427
- OLTP systems, 256
- separating data from, 116
- TextServer 3.0 system, 381
- types, 149
- unique indexes, 149, 427
- INIT_SQL_FILES**
 parameter, 632
- initialization parameters, 612**
- initializing**
- EXPLAIN PLAN command, 414-415
- SQL Trace parameters, 404-405
- Input/Output, see I/O**
- INSERT statement**
 defined, 612
 execution plan, displaying with EXPLAIN PLAN command, 414-416
- Inspect/SQL tool, 498**
- instance**
 archive logs, 134-135
 buffer cache, SQL statement, 597
 chained rows, SQL statement, 599
 checkpoints, 133-134
 data dictionary cache, SQL statement, 596-597
 defined, 4, 8, 612
 disk I/O
 SQL statement, 597-598
 storing data, 98
- free list contention, 136-137
 SQL statement, 602

latch contention, 130-131
SQL statement, 601
 library cache, SQL statement, 596
 memory
buffer cache, 107-108
operating system, 99
private PL/SQL area, 100
private SQL area, 100
shared pool, 100-106
sorts, 135-136
storing data, 98
 migrated rows, SQL statement, 599
 processes
background processes, 11-12
defined, 10
server (shadow) processes, 11
user (client) processes, 10
 recursive calls, SQL statement, 599
 redo log buffer contention, 130-131, 600-601
latch contention, 131-132
 rollback contention, SQL statement, 599-600
 rollback segments, 123-126
avoiding dynamic growth, 129
contention, 123-130
dynamic growth of, 600
extent size and number, 128-129
number of, 127-128
optimizing, 130
size, 128
 shared memory structure, 8
PGA (Program Global Area), 10
SGA (System Global Area), 9-10
 sorts, SQL statement, 601-602
 transactions, 25

INSTANCE_NUMBER
parameter, 641
integrity, 462
 constraints, 466
implementing at table creation, 466
implementing at table modification, 466-467
list of, 465
summary, 469
viewing, 467-469
 referential integrity
defined, 462, 614
tables, 463-464
 serial reads, 473
see also triggers

interfaces
 I/O, list of interfaces, 171
 raw device interface, UNIX, 198
IP/SQL statements, procedures, 24
ISM (Intimate Shared Memory), UNIX operating systems, 201
isolation, 56

J

JOB_QUEUE_INTERVAL
parameter, 632
JOB_QUEUE_KEEP_CONNECTIONS parameter, 632
JOB_QUEUE_PROCESSES parameter, 632
joins
 Cartesian products, 505, 609
 defined, 505, 612
 equijoins, 505, 611
 join conditions, 612
 nonequijoins, 613
 operations, hints, 484-485
 orders, hints, 484
 outer joins, 505, 507, 613
 self joins, 505-506, 615
 summary, 507
 tuning for response time or throughput, 507

K

KB (kilobyte), 30
kernels
 OS/2, 188
 Windows NT, 183
keys
 foreign keys, 611
 primary keys 614
keywords, triggers, 470

L

LABEL_CACHE_SIZE
parameter, 642
LANs (Local Area Networks), 612
latch contention, 130-131
 batch processing systems, 275
 BLOB systems, 332
 decision support systems, 295
 financial systems, 371
 office systems, 385
 OLTP systems, 255
 redo log buffer contention, 130-131
latch contention, 131-132

replicated systems, 376
 SQL statement, 601
 TextServer 3.0 systems, 381
 WebServer systems, 389
latency, 214
 rotational latency, 217
averages, 219

layers

HAL (Hardware Abstraction Layer), 611
logical layer
data blocks, 8
data dictionary, 5
data files, 5-6
defined, 5
extents, 8
schemas, 25
segments, 7
tablespaces, 5-6

- physical layer
 - control files*, 5
 - data files*, 5
 - defined*, 4
 - redo log files*, 5
- layouts**, 82
 - batch processing systems, 270-271
 - disk arrays*, 272-273
 - traditional disks*, 271-272
- BLOB system, 328
 - disk arrays*, 329-331
 - traditional disks*, 328-329
- data warehouses, 308-309
 - disk arrays*, 310-311
 - traditional disks*, 309-310
- decision support system, 291
 - disk arrays*, 292-293
 - traditional disks*, 291-292
- OLTP system, 250
 - disk arrays*, 252-253
 - traditional disks*, 250-252
- LCK (Parallel Server Lock processes)**, 12
- leaf blocks, indexes**, 149
- Level 1 cache, 206
- Level 2 cache, 206
- LGWR process (Log Writer)**, 11
- library cache**, 100-102, 450-452, 582
 - batch processing systems, 275
 - BLOB systems, 332
 - cache hits, 451
 - increasing*, 100-102
 - data warehouses, 313
 - decision support systems, 295
 - financial systems, 371
 - misses, 451
 - office systems, 384
 - OLTP systems, 254
- replicated systems, 376
- SQL statement, 596
- SQL Trace, statistics on library cache, 413
- stored functions or procedures
 - creating*, 456-457
 - replacing*, 457
- TextServer 3.0 systems, 381
- WebServer systems, 388
- LICENSE_MAX_SESSIONS parameter**, 632
- LICENSE_MAX_USERS parameter**, 633
- LICENSE_SESSIONS_WARNING parameter**, 633
- lightweight process**, 612
 - see also* processes; threads
- load balancing, disabling in UNIX**, 202
- load time, goals**, 44
- loads**
 - backup process, 353
 - batch processing systems, 267-268
 - BLOB systems, 324-325
 - data warehouses, 306-307
 - decision support systems, 288-289
 - OLTP systems, 247-248
 - parallel loading, 159
- Local Area Networks**, *see LANs*
- local transactions, distributed systems**, 378
- locking**
 - deadlocks, 510
 - defined, 508
 - DLM (Distributed Lock Manager), 162
 - explicit locking, 509
 - Parallel Server option, 342
 - DLM (Distributed Lock Manager)*, 341
 - PCM (Parallel Cache Management)*, 162, 342-343
- row-level locking, 508
- SELECT...FOR UPDATE statement**, 509
- serial reads, 508-509
- summary, 510
- table-level locking, 508
- log files, redo log files**, 614
- Log Writer**, *see LGWR*
- LOG_ARCHIVE_BUFFER_SIZE parameter**, 622
- LOG_ARCHIVE_BUFFERS parameter**, 622
- LOG_ARCHIVE_DEST parameter**, 633
- LOG_ARCHIVE_FORMAT parameter**, 633
- LOG_ARCHIVE_START parameter**, 634
- LOG_BUFFER parameter**, 622
- LOG_CHECKPOINT_INTERVAL parameter**, 622
- LOG_CHECKPOINT_TIMEOUT parameter**, 623
- LOG_CHECKPOINTS_TO_ALERT parameter**, 634
- LOG_FILES parameter**, 634
- LOG_SIMULTANEOUS_COPIES parameter**, 623
- LOG_SMALL_ENTRY_MAX_SIZE parameter**, 623
- logging**
 - archive logs, 134-135
 - batch processing systems*, 276
 - BLOB systems*, 332
 - data warehouses*, 314
 - decision support systems*, 295
 - financial systems*, 372
 - office systems*, 385
 - TextServer 3.0 systems*, 381
 - WebServer systems*, 389
 - events, triggers, 469
 - redo log files, 26-27
- logical disks**, 24, 612

logical layer
 data blocks, 8
 data dictionary, 5
 defined, 5
 extents, 8
 schema, 25
objects, 25
 segments, 7
 tablespaces
data files, 5-6
defined, 5-6
SYSTEM tablespace, 5-6

logical volumes
 defined, 226
 sizing, 238-239
 stripes, 226

logs, archive logs in OLTP systems, 255

M

main memory, 612
Massively Parallel Processor,
see MPP
master/slave architecture,
NetWare, 179
MAX_COMMIT_PROPAGATION_DELAY parameter, 641
MAX_DUMP_FILE_SIZE parameter, 405, 634
MAX_ENABLED_ROLES parameter, 634
MAX_ROLLBACK_SEGMENTS parameter, 634
MB (megabyte), 30
measurement conversions, 28
 binary storage units, 29-30
 powers of 10, 29
Media Server, 18
megabyte (MB), 30
memory
 architecture, 210
 batch processing
 systems, 276
 BLOB systems, 333
 buffers
block buffers, 22, 608
cache, 107-108, 608

clean buffers, 22, 608
defined, 22, 608
dirty buffers, 22, 608
redo log buffers, 608
cache
defined, 22, 608
hits, 609
 data warehouses, 314
decision support systems, 296
 main memory, defined, 612
 OLTP systems, 255
 operating systems, 170, 589
NetWare, 179-181
OS/2, 188-189
UNIX, 192-196, 531
Windows 3.11, 528
Windows 95, 530
Windows for Workgroups 3.11, 528
Windows NT, 185-186, 527
physical memory, defined, 24, 614
 private PL/SQL areas, 100
 private SQL areas, 100
 PSE (Page Size Extension)
 memory, 170
SGA
defined, 616
shared pool, 582
shared memory, 8
PGA (Program Global Area), 10
SGA (System Global Area), 9-10
shared pool, 100
cache miss, 101
data dictionary cache, 103-104
library cache, 100-102
shared session, 104-106
sorts, 135-136
storage capacity, 98
tuning operating system, 99
virtual memory
defined, 25, 617
paging, 211
swapping, 211

methodologies (performance tuning), 44
 analyzing results, 50
 defined, 41
 determining solution to problems, 48
testing solutions, 49
 examining systems for problems, 45-46
 finding cause of performance problems, 47-48
 goal setting for optimal performance, 48-50
 problem-solving methodology, flowchart, 604

microkernel architecture
 defined, 183, 612
Windows NT, 183-184

middleware
 defined, 556
 three-tiered system architecture, 557-558
tuning, 559
TM (Transaction Monitors)
client-side code, 560
defined, 559-561
server-side code, 560
tuning operating systems for, 561
 two-tiered system architecture, 556

migrating rows, 117-118
 SQL statement, 599

Mission Control, 496-497

MLS_LABEL_FORMAT parameter, 643

modes, current mode, 610

monitoring systems
 buying performance monitoring tools, 79
 third-party tools, 78
real-time monitors, 79
threshold monitors, 79-80

- tools, 74-75, 75
 EXPLAIN PLAN
 command, 77
 operating systems, 77-78
 Server Manager, 76
 SNMP agents, 76
 SQL Trace, 76-77
 *SQL*DBA Monitor*, 76
- MPP (Massively Parallel Processor)**, 209-210, 612
- MTS_DISPATCHERS parameter**, 637
- MTS_LISTENER_ADDRESS parameter**, 637
- MTS_MAX_DISPATCHERS parameter**, 637
- MTS_MAX_SERVERS parameter**, 637
- MTS_SERVERS parameter**, 638
- MTS_SERVICE parameter**, 638
- multiblock reads**, 152, 586
 batch processing systems, 275
 BLOB systems, 332
 data warehouses, 313
 decision support systems, 295
 financial systems, 372
 TextServer 3.0 systems, 381
 WebServer systems, 389
- multiblock writes**, 152-153, 586
 financial systems, 372
- multimedia systems**, performance criteria, 39
- multiprocessor systems**, 209
 defined, 612
 MPP (Massively Parallel Processor), 209-210, 612
 SMP (Symmetric Multiprocessor), 209, 615
- multithreaded servers**, list of parameters, 637-638
- N**
- National Language Support, list of parameters**, 643-644
- NetWare**, 178
 architecture, 178-179
 I/O subsystem, 182-183
 master/slave mode, 179
 memory, 179
 reducing unnecessary memory usage, 179-180
 SGA tuning, 180
 user capacity, 180-181
- networks, 181
 SPX/IPX, 181-182
 TCP/IP, 182
 tuning, 574-575
 summary of tuning guidelines, 182-183
- network frames**, *see* frames
- Network Interface Cards (NICs)**, 613
- network packets**, 613
 see also frames
- networks**
 backup process
 enhancements, 360
 bandwidth, defined, 608
 batch processing systems, 276
 enhancements, 280
 BLOB system
 enhancements, 335
 client, bottlenecks, 523
 collisions, 609
 data warehouse
 enhancements, 319
 decision support system
 enhancements, 300
 design considerations, 576
 bandwidth, 576-577
 bridges, 577
 hubs, 577
 routers, 577
 segmenting networks, 577
- design stage considerations, 84
- Ethernet, 611
hardware components, 566
 Ethernet networks, 566-568
 fiber optic networks, 569-570
 new technologies, 570
 Token Ring networks, 568-569
- OLTP systems, 255
 enhancements, 259
- operating systems
 NetWare, 181-182, 574-575
 OS/2, 190 575
 UNIX, 196-197, 531, 575
 Windows 3.11, 528-529
 Windows 95, 530
 Windows for Workgroups 3.11, 528-529
 Windows NT, 186, 575
- packets, testing size for backup process, 364
- performance criteria, client-server systems, 35
- protocols, 570
 SPX/IPX, 571
 TCP/IP, 571
- replicated system
 enhancements, 377
- software tuning, 574-576
- Token Ring, 617
tuning considerations, 85-86
 summary, 593
- New-Order transaction (TPC-C benchmark)**, 61
- nibbles**, 29
- NICs (Network Interface Cards)**, 613
- NLS_CURRENCY parameter**, 643
- NLS_DATE_FORMAT parameter**, 643
- NLS_DATE_LANGUAGE parameter**, 643

NLS_ISO_CURRENCY
 parameter, 643
NLS_LANGUAGE
 parameter, 644
NLS_NUMERIC_CHARACTERS
 parameter, 644
NLS_SORT parameter, 644
NLS_TERRITORY
 parameter, 644
No Data Protection mode
 (fault-tolerance), 233
NOAUDIT command,
 472-473
NOCACHE hint, 486
nonequijoins, 613
nonunique indexes, 149, 427
NOPARALLEL hint, 486
normal checkpoints, 26, 133
NOT NULL constraint, 465

O

objects, schema objects, 6, 25, 615
Objects for OLE:
 (development tool), 17
OCI (Oracle Call Interface),
 defined, 613
Office Server system,
 defined, 368
office system
 archive logs, 385
 block buffers, 384
 characteristics, 383
 checkpoints, 385
 contention, 385
 defined, 382-383
 design considerations, 384
 library cache, 384
 spin counts, 385
 summary of, 385-386
 tuning considerations,
 384-385
offline, 613
offline backups, *see cold*
 backups

OLTP (OnLine Transaction Processing)
 archive logs, 255
 block buffers, 254
 characteristics of
 system, 246
data access patterns,
 246-247
goals of optimally tuned
system, 248-249
loads, 247-248
summary, 249
 checkpoints, 255
 comparisons to batch
 processing systems, 265
 contention
latch contention, 255
rollback, 254
 defined, 27, 590, 613
 design considerations,
 249-250
 enhancements,
 256-257, 590
benchmarks, 262
clusters, 256-257
CPU, 257, 257-258
Etherplex board, 259
hash clusters, 256
I/O enhancements,
 258-259
indexes, 256
load testing, 260-261
network
enhancements, 259
Parallel Server
option, 257
testing operating system,
 261-262
testing RDBMS, 261
Transaction Monitors,
 259-260
verifying performance,
 260-261
 hardware
 considerations, 253
 library cache, 254
 physical data layout, 250
disk arrays, 252-253

traditional disks,
 250-252
 rollback segments, 128
 transactions, 65
 tuning considerations, 253
Oracle, 254-255
server operating system,
 255-256
online, 613
online tablespace backups,
see hot backups
OPEN_CURSORS
 parameter, 254, 634
OPEN_LINKS
 parameter, 635
OPEN_MOUNTS
 parameter, 643
operating systems
 asynchronous I/O, 589
 batch processing systems
testing, 282, 363-365
tuning considerations,
 276-277
BLOB systems
testing, 336
tuning
considerations, 333
 bottlenecks, finding, 93
 data warehouses
testing, 320
tuning considerations,
 314-315
 decision support systems,
 295-296
testing, 301
I/O, 170-172
 memory, 170
 microkernel, 612
 monitoring performance,
 tools, 77-78
NetWare, 178
architecture, 178-179
I/O subsystem, 182-183
master/slave mode, 179
memory, 179-181
networks, 181-182
summary of tuning
guidelines, 182-183

- new features
 cache affinity, 174, 589
 post-wait semaphore, 173, 589
 preemption, 173-174
 scheduling processes, 173-174
- OLTP systems, tuning considerations, 255-256
- Oracle, network tuning, 575-576
- OS/2, 188
 architecture, 188
 I/O subsystem, 190-191
 memory, 188-189
 networks, 190, 575
 summary of tuning guidelines, 190-191
- processes, 169
- scheduling parameters, 589
- striping, 113-114
- tuning considerations, 94, 99
 goals of optimally tuned systems, 168-169, 589
- UNIX, 530-531
 architecture, 192
 benchmarking new features, 200-203
 development of, 191
 disabling preemptive scheduling, 201
 I/O subsystem, 197-200
 ISM (Intimate Shared Memory), 201-203
 load balancing, 202
 memory, 192-196, 531
 networks, 196-197, 531, 575
 new features, 201-203
 post-wait semaphores, 201-203
- Windows 3.1
 memory, 528
 networks, 528-529
- Windows 95
 32-bit support, 529-530
 memory, 530
- networks*, 530
 Oracle support, 530
- Windows for Workgroups 3.11
 memory, 528
 networks, 528-529
- Windows NT, 183
 16-bit applications, 527
 architecture, 183-184
 defined, 527
 I/O performance, 528
 I/O subsystem, 186-187
 memory, 185-186, 527
 networks, 186, 575
 summary of tuning guidelines, 187
 threads, 184
- optimal backups, enhancements**, 591
- optimization techniques**
 ANALYZE command, 441-442
 checking structural integrity, 444
 data dictionary statistics, 445-447
 gathering chained-row statistics, 444-445
 gathering statistics, 442-443
 choosing, 438-439
 cost-based approach, 434, 441, 610
 defined, 433, 438-440, 613
 flowcharts (Oracle optimizer), 605
 hints, 447, 478
 ALL_ROWS hint, 479
 CHOOSE hint, 479
 FIRST_ROWS hint, 479-480
 RULE hint, 480-481
 rule-based approach, 433-434, 440, 614
- OPTIMIZER_MODE parameter**, 623
- Oracle, network tuning**, 575-576
- Oracle Call Interface (OCI)**, 613
- Oracle Corporation**, 3
- Oracle Financials**, *see financial systems*
- Oracle Mission Control**, 496-497
- Oracle Office**, 17-18
- Oracle Office Server**, *see Office Server system*
- Oracle optimizer, flowchart**, 605
- Oracle Power Objects**, 495
 see also Power Objects
- Oracle TextServer 3.0**, *see TextServer 3.0 system*
- Oracle Trace**, 497
- Oracle WebServer**, *see WebServer system*
- Order-Status transaction (TPC-C benchmark)**, 62
- ORDERED hint**, 484
- OS/2, 188**
 architecture, 188
 I/O subsystem, 190-191
 memory, 188
 reducing unnecessary memory usage, 189
 SGA tuning, 189
 user capacity, 189
- networks, 190
 tuning, 575
 summary of tuning guidelines, 190-191
- OS_AUTHENT_PREFIX parameter**, 642
- OS_ROLES parameter**, 642
- outer joins**, 505, 507, 613
- overhead, I/O overhead**, 117
 chaining rows, 117-118
 checking for recursive calls, 118-119
 dynamic extensions, 118-119
 migrating rows, 117-118
- PCTFREE command option, 119-122

-
- PCTUSED command
option, 119-122
reduction techniques, 122
- P**
- packages**, **432-433, 457-458**
advantages, 458
body, 458
declarations, 458
defined, 24, 450, 613
program units, defined, 614
RDBMS_ALERT
 package, 470
RDBMS_OUTPUT
 package, 456
syntax, 458
see also functions;
procedures
- packets (network packets)**, **613**
Ethernet, 567
testing for backup process, 364
- Page Size Extension (PSE) memory**, **170**
- paging**, **24, 211, 613**
see also swapping
- Parallel Cache Management**,
see PCM
- PARALLEL hint**, **485-486**
- Parallel Query option (add-on)**, **586-587, 613**
batch processing systems, 277-278
BLOB systems, 333
data warehouses, 316
decision support systems, 297
defined, 15, 153
Direct Write Sorts option, 158
financial systems, 372
hints
 CACHE, 485
 NOCACHE, 486
 NOPARALLEL, 486
- PARALLEL_DEFAULT_MAX_INSTANCES parameter**, **347, 641**
- PARALLEL_DEFAULT_MAX_SCANS parameter**, **587, 626**
- PARALLEL_DEFAULT_SCANSIZE parameter**, **587, 627**
- PARALLEL_MAX_SERVERS parameter**, **587, 627**
- PARALLEL_MIN_SERVERS parameter**, **627**
- PARALLEL_SERVER_IDLE_TIME parameter**, **627**
- parameters**
analysis tools, list of parameters, 627-629
AUDIT_TRAIL, 641
AUTO_MOUNTING, 642
BACKGROUND_DUMP_DEST, 629
BLANK_TRIMMING, 629
CACHE_SIZE_THRESHOLD, 639
CHECKPOINT_PROCESS, 629
CLEANUP_ROLLBACK_ENTRIES, 630
CLOSE_CACHED_OPEN_CURSORS, 630
COMMIT_POINT_STRENGTH, 638
COMPATABLE, 630
COMPATIBLE_NO_RECOVERY, 630
CONTROL_FILES, 630
CREATE SEQUENCE command, 502-503
DB_BLOCK_BUFFERS, 620
DB_BLOCK_CHECKPOINT_BATCH, 620-621
DB_BLOCK_CHECKSUM, 628
DB_BLOCK_LRU_EXTENDED_STATISTICS, 628
DB_BLOCK_LRU_STATISTICS, 628
DB_BLOCK_SIZE, 621
DB_DOMAIN, 630
DB_FILE_MULTIBLOCK_READ_COUNT, 621

DB_FILE_SIMULTANEOUS_WRITE	621	INSTANCE_NUMBER,	641	MTS_DISPATCHERS,	637
DB_FILES	631	JOB_QUEUE_INTERVAL,	632	MTS_LISTENER_ADDRESS,	637
DB_LOG_CHECKSUM	628	JOB_QUEUE_KEEP_CONNECTIONS	632	MTS_MAX_DISPATCHERS,	637
DB_MOUNT_MODE	642	JOB_QUEUE_PROCESSES,	632	MTS_MAX_SERVERS,	637
DB_NAME	631	LABEL_CACHE_SIZE,	642	MTS_SERVERS,	638
DBLINK_ENCRYPT_LOGIN	631	LICENSE_MAX_SESSIONS,	632	MTS_SERVICE,	638
DISCRETE_TRANSACTIONS_ENABLED	621	LICENSE_MAX_USERS,	633	multithreaded servers, list of parameters	637-638
distributed option, list of parameters	638-639	LICENSE_SESSIONS_WARNING	633	National Language Support, list of parameters	643, 644
DISTRIBUTED_LOCK_TIME-OUT	638	LOG_ARCHIVE_BUFFER_SIZE	622	NLS_CURRENCY,	643
DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME	638	LOG_ARCHIVE_BUFFERS,	622	NLS_DATE_FORMAT,	643
DISTRIBUTED_TRANSACTIONS	639	LOG_ARCHIVE_DEST,	633	NLS_DATE_LANGUAGE,	643
DML_LOCKS	622	LOG_ARCHIVE_FORMAT,	633	NLS_ISO_CURRENCY,	643
ENQUEUE_RESOURCES	631	LOG_ARCHIVE_START,	634	NLS_LANGUAGE,	644
EVENT	628	LOG_BUFFER	622	NLS_NUMERIC_CHARACTERS,	644
FIXED_DATE	628	LOG_CHECKPOINT_TIMEOUT	623	NLS_SORT,	644
functions	454	LOG_CHECKPOINTS_TO_ALERT	634	NLS_TERRITORY,	644
GC_DB_LOCKS	639	LOG_CHECKPOINT_INTERVAL	622	OLTP systems,	254
GC_FILES_TO_LOCKS	640	LOG_FILES	634	OPEN_CURSORS,	634
GC_LCK_PROCS	640	LOG_SIMULTANEOUS_COPIES	623	OPEN_LINKS,	635
GC_ROLLBACK_LOCKS	640	LOG_SMALL_ENTRY_MAX_SIZE	623	OPEN_MOUNTS,	643
GC_ROLLBACK_SEGMENTS	640	MAX_COMMIT_PROPAGATION_DELAY	641	OPTIMIZER_MODE,	623
GC_SAVE_ROLLBACK_LOCKS	640	MAX_DUMP_FILE_SIZE	634	OS_AUTHENT_PREFIX,	642
GC_SEGMENTS	640	MAX_ENABLED_ROLES	634	OS_ROLES,	642
GC_TABLESPACES	640	MAX_ROLLBACK_SEGMENTS	634	Parallel Query option, list of parameters	626-627
general, list of parameters	629-637	MLS_LABEL_FORMAT	643	Parallel Server option, list of parameters	639-641
GLOBAL_NAMES	632			PARALLEL_DEFAULT_MAX_INSTANCES	641
IFILE	632			PARALLEL_DEFAULT_MAX_SCANS	626
INIT_SQL_FILES	632			PARALLEL_DEFAULT_SCAN-SIZE	627
initialization parameters	612			PARALLEL_MAX_SERVERS	627

- PARALLEL_MIN_SERVERS, 627
PARALLEL_SERVER_IDLE_TIME, 627
performance, list of parameters, 620-626
PRE_PAGE_SGA, 623
PROCESSES, 635
RECOVERY_PARALLELISM, 627
REMOTE_LOGIN_PASSWORDFILE, 635
REMOTE_OS_AUTHENT, 639
REMOTE_OS_ROLES, 639
ROLLBACK_SEGMENTS, 624
ROW_CACHE_CURATORS, 624
ROW_LOCKING, 624
SCO UNIX, asynchronous I/O, 199
security, list of parameters, 641-642
SEQUENCE_CACHE_ENTRIES, 624
SEQUENCE_CACHE_HASH_BUCKETS, 624
SERIALIZABLE, 624-625
SESSION_CACHED_CURSOR, 625
SESSIONS, 635
SHARED_POOL_SIZE, 625
SINGLE_PROCESS, 636
SMALL_TABLE_THRESHOLD, 625
SNAPSHOT_REFRESH_INTERVAL, 635
SNAPSHOT_REFRESH_KEEP_CONNECTION, 635
SNAPSHOT_REFRESH_PROCESS, 636
SORT_AREA_RETAINED_SIZE, 625
SORT_AREA_SIZE, 626
SORT_SPACEMAP_SIZE, 626
SQL Trace, 404-405
 interpreting, 409
 TKPROF program, 407-409
SQL_TRACE, 629
SQL92_SECURITY, 642
TEMPORARY_TABLE_LOCKS, 636
THREAD, 641
TIMED_STATISTICS, 629
TRANSACTIONS, 636
TRANSACTIONS_PER_ROLLBACK_SEGMENT, 636
Trusted Oracle7 option, list of parameters, 642-643
UNIX
 asynchronous I/O, 198
 SHMMAX, 194
 SHMSEG, 194
UnixWare, asynchronous I/O, 199
USER_DUMP_DEST, 637
- parsing SQL statements, 399-400**
- Partial Data Protection mode (fault-tolerance), 234-235**
- Patrol tool, 498-499**
- Payment transaction (TPC-C benchmark), 62**
- PCM locks (Parallel Cache Management), 162**
 Parallel Server option, 342-343
- PCTFREE command option, 119-122**
- PCTUSED command option, 119-122**
- performance criteria**
 batch-processing systems, 38
 checklist, 39
 client/server systems, 32-33
 checklist, 35-36
 client component, 33
- network*, 35
server component, 33-35
determining problems, 46
multimedia systems, 39
terminal-based systems (host-based), 36
 checklist, 37-38
 database, 36-37
 front-end applications, 36
- performance monitoring tools, 73-75**
buying, 79
EXPLAIN PLAN
 command, 77
operating systems, 77-78
Server Manager, 76
SNMP agents, 76
SQL Trace, 76-77
SQL*DBA Monitor, 76
third-party tools, 78-79
 real-time monitors, 79
 threshold monitors, 79-80
- performance problems**
finding, 92-93
methodology
 analyzing results, 50
 examining systems for, 45-46
 finding cause of, 47-48
 goal setting for optimal performance, 48-50
 solutions for, 48-49
- persistent areas (private SQL area), 100**
- Personal Oracle for Windows, 16**
- PGA, 10**
- physical data layout**
batch processing systems, 270-271
 disk arrays, 272-273
 traditional disks, 271-272
- BLOB systems
 disk arrays, 329-331
 traditional disks, 328-329

- data warehouses, 308-309
 disk arrays, 310-311
 traditional disks,
 309-310
- decision support systems
 disk arrays, 292-293
 traditional disks,
 291-292
- OLTP (OnLine Transaction Processing), 250
 disk arrays, 252-253
 traditional disks,
 250-252
- physical layer**
 control files, 5
 data files, 5
 defined, 4
 redo log files, 5
- physical memory**, 24, 614
- PL/SQL**
 defined, 14, 614
 functions, 24, 454-455
 private PL/SQL areas, 100
 procedures, 454-455
 properties, 454-455
- PL/SQL Interpreter (Procedure Builder tool)**, 542
- platters (disk drive)**, 214
 cylinders, 215
 tracks, 214
- PMON (Process Monitor)**, 11
- post-wait semaphore feature**, 173
 operating systems, 589
 UNIX, 201
- Power Objects (development tool)**, 17, 495
 classes, 545-546
 features, 544-546
 Forms Designer tool,
 544-545
- PowerBuilder**, 495, 552
- PRE_PAGE_SGA parameter**, 623
- Precise/SQL suite**, 498
- preempting processes**, 173-174
- preemptive scheduling**, UNIX
 cache affinity, 202
 disabling, 201
- PRIMARY constraint**, 465
- primary keys**, 614
- private SQL area (memory)**, 10
 persistent areas, 100
 runtime areas, 100
- problem-solving methodology**, flowchart, 604
- Procedural option**, *see* PL/SQL
- Procedure Builder tool (Designer/2000)**, 537, 542-544
- procedures**, 432-433
 body, 453
 comparisons to functions, 452
 declarations, 453
 defined, 24, 452-453, 614
 EXCEPTION, 455
 packages, 24
 PL/SQL language, 454-455
 RDBMS_OUTPUT
 package, 456
 RDBMS_OUTPUT.DISABLE, 456
 RDBMS_OUTPUT.ENABLE, 456
 RDBMS_OUTPUT.GET_LINE, 456
 RDBMS_OUTPUT.GET_LINES, 456
 RDBMS_OUTPUT.PUT, 456
 RDBMS_OUTPUT.PUT_LINE, 456
- READ_CLIENT_INFO**, 417
- READ_MODULE, 417
- SET_ACTION, 417
- SET_CLIENT_INFO, 417
- SET_MODULE, 417
- stored procedures
 creating, 456-457
 defined, 7, 452
 program units, 614
 properties, 452
 replacing, 457
- syntax, 453
 see also functions
- Process Modeller tool**, 491
- Process Monitor (PMON)**, 11
- PROCESSES parameter**, 635
- processes**
 ARCH, 26-27
 background
 ARCH (Archiver), 12
 CKPT (Checkpoint), 11, 26
 DBWR (Database Writer), 11
 Dnnn (Dispatcher), 12
 LCK (Parallel Server Lock), 12
 LGWR (Log Writer), 11
 PMON (Process Monitor), 11
 RECO (Recovery), 12
 SMON (System Monitor), 12
 calculating number for system, 169
 defined, 10
 operating systems, 169
 prioritizing, 169
 scheduling, 173-174
 server (shadow), 11
 user (client), 10
 see also lightweight processes; threads; traditional processes

processing

array processing, 510
queries, 400
SQL statements, 397
bind variables, 401
cursor creation, 398
executing statements, 401
fetching returned rows, 401-402
parallelization, 401
parsing statement, 399-400
query processing, 400
summary, 402
transactions, 395-396

processors

CPU
backup process
enhancements, 359
batch processing system
enhancements, 279
BLOB system
enhancements, 334
cache, 210
CISC (Complex Instruction Set Computer), 207-208
data warehouse
enhancements, 317
decision support system
enhancements, 298-299
OLTP enhancements, 257-258
replicated system
enhancements, 377
RISC (Reduced Instruction Set Computer), 207-208
multiprocessor systems, 209
MPP (Massively Parallel Processor), 209-210
SMP (Symmetric Multiprocessor), 209

products

applications, 17
ConText, 18
Media Server, 18
Oracle Financial Applications, 17

Oracle Office, 17-18

Oracle WebServer, 18
development tools, 16-17
Oracle RDBMS, 13
Distributed option, 14
Parallel Query option, 15
Parallel Server option, 14
PL/SQL, 14
Server Manager, 15
*SQL*Net*, 14
Symmetric Replication, 15
Trusted Oracle, 15

Personal Oracle for

Windows, 16

services

consulting, 18
education, 19
support, 19

Workgroup Server, 15-16

Program Unit Editor (Procedure Builder tool), 542

program units, 24, 614

protocols, 570-571

SPX/IPX, 571, 615
TCP/IP (Transmission Control Protocol/Internet Protocol), 571, 616

PSE (Page Size Extension) memory, 170

PUSH_SUBQ hint, 486

Q

queries

ad-hoc, 22, 608
business support systems, 611
component queries, 609
compound queries, 609
decision support systems, 28
defined, 614

joins

defined, 612
join conditions, 612
nonequijoins, 613
outer joins, 613
selfjoins, 615

Parallel Query option, 153

degree of parallelism, 156-158

Direct Write Sorts option, 158

I/O configuration, 156

parallel index creation, 159

parallel loading, 159

parallel query operation, 153-158

parallel recovery, 160-161

processing, 400

rollback segments, 128
subqueries, 609, 616

query parameter, SQL Trace, 409

queueing, 218

R

RAID (Redundant Array of Inexpensive Disks), 229

comparing RAID levels, 240-241

configuration issues, 235-236

distributing random I/Os, 237-238

isolating sequential I/Os, 236-237

sizing logical volume, 238-239

RAID-0, 230

RAID-1, 230-231

RAID-2, 231

RAID-3, 231

RAID-4, 232

RAID-5, 232

summarizing RAID levels, 232

- RAM (Random Access Memory), 612**
random I/Os, 220
defined, 24, 111, 598, 614
disk arrays, distributing random I/Os, 237-238
disk rates, 598
spreading out, 590, 598
striping data, 112-113
 hardware, 114-115
 operating system, 113-114
 options, 115-116
- raw device interface, UNIX, 198**
- RDBMS (Relational Database Management Systems)**
databases, 4
defined, 4
instance, 4
products, 13
 Distributed option, 14
 Parallel Query option, 15
 Parallel Server option, 14
 PL/SQL, 14
 Server Manager, 15
 *SQL*Net, 14*
 Symmetric Replication, 15
 Trusted Oracle, 15
- RDBMS_ALERT**
package, 470
- RDBMS_OUTPUT**
package, 456
- RDBMS_OUTPUT.DISABLE**
procedure, 456
- RDBMS_OUTPUT.ENABLE**
procedure, 456
- RDBMS_OUTPUT.GET_LINE**
procedure, 456
- RDBMS_OUTPUT.PUT**
procedure, 456
- RDBMS_OUTPUT.PUT_LINE**
procedure, 456
- read consistency, 24, 614**
- Read Lock mode (Parallel Server option), 342**
- read-only snapshots, 374**
- read/write caches, disk arrays, 229**
- READ_CLIENT_INFO procedure, 417**
- READ_MODULE procedure, 417**
- real-time monitoring tools, 79**
- recovery process (RECO), 12, 351-352**
- RECOVERY_PARALLELISM parameter, 587, 627**
- recursive calls**
 checking, 118
 defined, 118, 614
 SQL statement, 599
- redo log buffer, 10, 608**
contention
 latch contention, 131-132
 SQL statement, 600-601
- redo log files, 5, 26-27, 614**
- Reduced Instruction Set Computer, *see* RISC processors**
- Redundant Array of Inexpensive Disks, *see* RAID**
- referential integrity**
defined, 462, 614
tables, 463-464
- registering applications, 417**
- Relational Database Management System, *see* RDBMS**
- remote transactions, distributed systems, 378**
- REMOTE_LOGIN_PASSWORDFILE parameter, 635**
- REMOTE_OS_AUTHENT parameter, 639**
- REMOTE_OS_ROLES parameter, 639**
- replicated systems**
 block buffers, 376
 characteristics, 374-375
 contention, 376
- defined, 368, 374
design considerations, 375-377
 goals of optimally tuned systems, 375
library cache, 376
read-only snapshots, 374
snapshots, 615
summary of, 377
symmetric replication, 15, 374
updatable, 374
- replication, 614**
- Reports Designer tool**
Designer/2000, 536, 538-540
Oracle Power Objects, 545
- Reports Generator tool, 492**
- ReportSmith, 548-549**
- response time**
analyzing problems, 47-48
defined, 42
goals, 42-43
performance criteria, terminal-based systems, 37
- RETURN function, 454, 455**
- RISC processors (Reduced Instruction Set Computer), 208**
- rollback contention**
defined, 123-126, 130, 614
financial systems, 371
office systems, 385
OLTP systems, 254
replicated systems, 376
SQL statements, 599-600
WebServer systems, 389
- rollback segments**
avoiding dynamic growth of, 129
batch processing systems, 275
BLOB systems, 332
data warehouses, 313
decision support systems, 295
defined, 8, 614
dynamic growth of, 600

extent size and number, 128-129
 financial systems, 371
 number of, 127-128
 size, 128
 TextServer 3.0
 systems, 381
 transaction tables, 124

ROLLBACK_SEGMENTS
 parameter, 624
rotational latency, 217
 averages, 219

routers, network design
 considerations, 577

row cache, *see* data dictionary, cache

row-level locking, 508

ROW_CACHE_CURSORS
 parameter, 624

ROW_LOCKING
 parameter, 624

ROWID hint, 483

rows
 chaining, 117-118
 SQL statement, 599
 migrating, 117-118
 SQL statement, 599

rows parameter, SQL Trace, 409

RULE hint, 480-481

rule-based optimization
 approach, 433-434, 440
 choosing, 438-439
 defined, 614
 hints, 447

runtime areas (private SQL area), 100

S

S-Designer tool, 493

scalability, 24, 615

Scheduler
 batch processing systems, 276
 BLOB systems, 333
 data warehouses, 314
 decision support systems, 296
 OLTP systems, 255

scheduling
 operating systems
 parameters, 589
 UNIX options, 201-203
 processes, 173-174

schemas
 defined, 6-7, 25, 615
 objects, 6, 25, 615

SCO UNIX
 buffer cache, tuning, 193
 networks, 196-197
 parameters
 asynchronous I/O, 199
 shared memory, 195

scripts, CD-ROM, 646-648

searches, *see* TextServer 3.0 systems

security, list of parameters, 641-642

seek time, 216-217
 averages, 219

segments
 backups, 360-362
 data segments, 8
 defined, 7, 118, 615
 index segments, 8
 networks, 577
 rollback segments, 123-126
 avoiding dynamic growth of, 129
 batch processing systems, 275
 BLOB systems, 332
 contention, 123-130
 data warehouses, 313
 decision support systems, 295
 defined, 8
 dynamic growth of, 600
 extent size and number, 128-129
 financial systems, 371
 number of, 127-128
 optimizing, 130
 size, 128
 SQL statement, 599-600
 TextServer 3.0 systems, 381
 transaction tables, 124

temporary segments, 8
see also data blocks; extents

SELECT statement, displaying execution plan, 414-416

SELECT...FOR UPDATE statement, 509

self joins, 505-506, 615

SEQUENCE_CACHE_ENTRIES parameter, 624

SEQUENCE_CACHE_HASH_BUCKETS parameter, 624

sequences
 defined, 615
 program units, 614
 table sequences
 creating, 502-503
 defined, 502
 generating primary key values, 504-505
 generating sequence values, 503-504
 summary, 504-505
 tuning
 considerations, 503

sequential I/O
 defined, 25, 111, 220-222, 598, 615
 disk arrays, isolating
 sequential I/Os, 236-237
 disk rates, 598
 isolating, 112, 590, 598

serial reads, 473
 joins, 508-509

SERIALIZABLE parameter, 624-625

Server Generator tool, 492

server interconnect, 162

Server Manager, 15, 76, 615

server processes (shadow processes), 11

servers
 multithreaded servers, parameters, 637-638
 performance criteria, 33-35
 tuning performance, 85

services (Oracle)
consulting, 18
education, 19
support, 19

session information
(PGA), 10

SESSION_CACHED_CURSORS
parameter, 625

SESSIONS parameter, 635

sessions, 615

SET_ACTION
procedure, 417

SET_CLIENT_INFO
procedure, 417

SET_MODULE procedure,
417

SGA (System Global Area)
buffer cache, 9
defined, 9-10, 25, 615
redo log buffer, 10
shared pool, 10, 582
buffer cache, 583
data dictionary cache, 582
library cache, 582
shared session information, 583

shadow processes, *see server processes*

shared memory, 8

PGA (Program Global Area), 10

SGA (System Global Area)
buffer cache, 9
redo log buffer, 10
shared pool, 10

UNIX, parameters, 194

shared pool
buffer cache, 583
cache hit ratio, 107-108
cache miss, 101
data dictionary cache, 103-104, 582
defined, 10, 100, 582
library cache, 100-102, 582
increasing cache hits, 100-102

shared session area, 104-106, 583

SHARED_POOL_SIZE
parameter, 625

shareware, middleware
defined, 556
three-tiered system, 557-559
two-tiered system architecture, 556

shipping industry systems, performance criteria, 39

SHMMAX parameter, 194

SHMSEG parameter, 194

Simple Network Management Protocol (SNMP), agents, 76

simple statements, 615

SINGLE_PROCESS
parameter, 636

SMALL_TABLE_THRESHOLD
parameter, 625

SMON (System Monitor), 12

SMP (Symmetric Multiprocessor), 209, 615

SNAPSHOT_REFRESH_INTERVAL parameter, 635

SNAPSHOT_REFRESH_KEEP_CONNECTION parameter, 635

SNAPSHOT_REFRESH_PROCESS parameter, 636

snapshots
defined, 615
read-only snapshots, 374
updatable snapshots, 374

SNMP (Simple Network Management Protocol), agents, 76

software
backup process
cold (offline) backups, 355
hot (online) backups, 357-358
testing, 364-365

networks, 574
NetWare, 574-575
Oracle tuning, 575-576
OS/2, 575
UNIX, 575
Windows NT, 575

software disk arrays, 227

Solaris
buffer cache, tuning, 193
networks, 197
shared memory, parameters, 194-195

SORT_AREA_RETAINED_SIZE
parameter, 625

SORT_AREA_SIZE
parameter, 626

SORT_SPACEMAP_SIZE
parameter, 626

sorts
direct write sorts, 143, 158
batch processing systems, 277
data warehouses, 315
decision support systems, 296
memory, 135-136
SQL statement, 601-602

Source Code Control (Procedure Builder tool), 542

spin counts, 164, 588

financial systems, 371
office systems, 385
WebServer systems, 388

SPX/IPX protocol
defined, 571, 615
operating systems, NetWare, 181-182

SQL (Structured Query Language)
cursors, 610
queries
defined, 614
joins, 612
subqueries, 609, 616
scripts on CD-ROM, 646-648

- statements
- analyzing effects of*, 424-425
 - analyzing with EXPLAIN PLAN command*, 424
 - analyzing with SQL Trace*, 423-424
 - buffer cache*, 597
 - chained rows*, 599
 - characteristics of optimized statements*, 394
 - complex statements*, 609
 - data dictionary cache*, 596-597
 - determining dynamic rollback growth*, 600
 - executing*, 401
 - flowchart of statement processing*, 605
 - free list contention*, 602
 - functions*, 24
 - hints*, 394
 - identifying badly formed statements*, 394-395
 - I/O usage*, 597-598
 - latch contention*, 601
 - library cache*, 596
 - migrated rows*, 599
 - packages*, 394
 - procedures*, 24, 394, 614
 - processing*, 397-402
 - recursive calls*, 599, 614
 - redo log buffer contention*, 600-601
 - rollback contention*, 599-600
 - simple statements*, 615
 - sort performance*, 601-602
 - transaction processing*, 395-396
 - tuning*, 422-424
 - UPDATE statement*, 617
 - tuning private SQL areas*, 100
 - SQL Coder tool**, 493-494
- SQL DBA, *see* Server Manager**
- SQL Trace feature**
- analyzing SQL statements*, 423-424
 - defined*, 76-77, 404
 - initialization parameters*, 404-405
 - disabling command*, 410
 - for an instance*, 406
 - per-session basis*, 405-406
 - elapsed time*, 412
 - enabling command*, 409
 - for an instance*, 406
 - per-session basis*, 405
 - functionality*, 406
 - interpreting information*, 409-414
 - library cache statistics*, 413
 - summary of*, 414
 - TKPROF program**
 - functionality*, 407-409
 - output file*, 410-414
 - parameters*, 407-409
- SQL Windows**, 495, 550-552
- SQL*DBA Monitor**, 76
- SQL*Loader**, 616
- SQL*Net**, 14, 616
- SQL*Plus**, 616
- SQL/DBA**, 616
- SQL_TRACE parameter**, 629
- SQL92_SECURITY parameter**, 642
- stack space (PGA)**, 10
- standard benchmarks**
- batch processing systems*, 282-283
 - BLOB systems*, 336-337
 - data warehouses*, 320-321
 - decision support systems*, 301-302
 - defined*, 51-53
 - OLTP systems*, 262
 - TPC (Transaction Processing Performance Council)*, 53-54
- ACID tests (Atomicity, Consistency, Isolation, and Durability)*, 55-58
- interpreting spreadsheet of results*, 67-69
- members*, 53-54
- publishing benchmarks*, 57-58
- results of benchmarks*, 56-58, 67-69
- rules*, 55
- steering committee*, 54
- Technical Advisory Board (TAB)*, 54
- TPC-A benchmark*, 59-60
- TPC-B benchmark*, 60-61
- TPC-C benchmark*, 61-63
- TPC-C/S (client/server) benchmark*, 66-67
- TPC-D benchmark*, 63-64
- TPC-E benchmark*, 64-66
- statements**
- analyzing effects of*, 424-425
 - cache*
 - buffer cache*, 597
 - data dictionary cache*, 596-597
 - library cache*, 596
 - characteristics of optimized statements*, 394
 - complex*, 609
 - contention*
 - free list*, 602
 - latch*, 601
 - redo log buffer*, 600-601
 - rollback*, 599-600
- DELETE**, 611
- determining dynamic rollback growth*, 600

EXPLAIN PLAN
 analyzing statement execution, 424
 defined, 414
 extracting results of, 416
 initialization, 414-415
 invoking, 415-416
flowchart of statement processing, 605
functions, 24
hints, 394
 errors, 477-478
 examples, 477-478
 implementing, 476-478
 multiple, 478
 optimization techniques, 478-481
 syntax, 477
identifying badly formed statements, 394-395
INSERT, 612
I/O usage, 597-598
joins, equijoins, 611
optimizing, 438
packages, 394
procedures, 24, 394, 614
processing, 397
 bind variables, 401
 cursor creation, 398
 executing statements, 401
 fetching returned rows, 401-402
 parallelization, 401
 query processing, 400
 statement parsing, 399-400
 summary, 402
recursive calls, 599, 614
rows, chained or migrated, 599
simple statements, 615
sort performance, 601-602
SQL Trace feature
 analyzing statements, 423-424
 defined, 404

disabling, 405-406, 410
elapsed time, 412
enabling, 405-406, 409
functionality, 406
initialization parameters, 404-405
interpreting information, 409-414
library cache
 statistics, 413
summary, 414
TKPROF output file, 410-414
TKPROF program, 407-409
transaction processing, 395-396
tuning, 422-424
UPDATE, 617
steering committee
(TPC), 54
Stock-Level transaction (TPC-C benchmark), 62
stored functions
 creating, 456-457
 defined, 452
 properties, 452
 replacing, 457
 see also functions
stored procedures
 creating, 456-457
 defined, 7, 452
 program units, 614
 properties, 452
 replacing, 457
 see also procedures
Stored Program Unit Editor (Procedure Builder tool), 542
storing tables, *see clusters*
streaming, 616
stripes, 226
striping data
 defined, 112
 hardware, 114-115
 operating system, 113-114
 options, 115-116
 Oracle, 112-113
striping factor, 226
Structured Query Language,
 see SQL
subnetting networks, 576
subqueries, 609, 616
suites, Oracle Office, 17-18
support services, 19
swapping, 25, 211, 616
 see also paging
Symmetric Multiprocessor,
 see SMP
Symmetric Replication, 15, 374
synchronous I/O, 171-172
synonyms, 616
syntax
 functions, 453
 hints, 477
 packages, 458
 procedures, 453
 triggers, 470
System Global Area, *see SGA*
system memory
 architecture, 210
 virtual memory, 211
 defined, 210
 speed, 206
System Monitor, *see SMON*
SYSTEM tablespace, 5
systems, limitations, 95
Systems Designer tool, 491-492
Systems Modeller tool, 491

T

table sequences
 creating, 502-503
 defined, 502
 generating primary key values, 504-505
 generating sequence values, 503-504
 summary, 504-505
 tuning considerations, 503
table-level locking, 508

- tables**
 Cartesian products, 609
 clusters, 6, 609
 columns, 6
 data dictionary, 610
 defined, 6, 616
 dynamic performance tables
 defined, 24, 611
 views (table of), 74-75
 indexes, choosing tables to index, 427-428
 integrity, referential, 463-464
 transaction tables, rollback segments, 124
V\$ROLLSTAT, rollback segment data, 129
V\$ROWCACHE, data dictionary statistics, 103
V\$SYSSTAT, buffer cache statistics, 107
V\$WAITSTAT, rollback segment data, 127
 views, data in, 446
- tablespaces**
 data files, 5-6
 defined, 5-6, 616
 fragmentation, 144-146
 SYSTEM tablespace, 5-6
- tape drive, testing block size for backup process**, 363
- tapes, streaming**, 616
- TB (terabyte)**, 30
- TCP/IP (Transmission Control Protocol/Internet Protocol)**, 571
 defined, 616
 operating systems, NetWare, 182
- temporary segments**, 8
- TEMPORARY_TABLE_LOCKS parameter**, 636
- terabyte (TB)**, 30
- terminal-based systems (host-based systems)**, performance criteria, 36
 checklist, 37-38
 database, 36-37
 front-end application, 36
- testing performance, see benchmarks**
- testing stage**, 84
 clients, 85
 networks, 85-86
 servers, 85
- TextServer 3.0 system**
 archive logs, 381
 block buffers and size, 381
 characteristics, 379-380
 checkpoints, 381
 clusters, 381
 contention, latch contention, 381
 defined, 368, 379
 design considerations, 380-381
 enhancements, 381-382
 hash clusters, 381
 indexes, 381
 library cache, 381
 multiblock reads, 381
 Parallel Query, 381
 segments, rollback, 381
 summary, 382
 tuning considerations, 381
- third-party tools**
 analysis tools, 497
AdHawk, 498
DATA-XPERT, 498
DBGENERAL, 498
Patrol, 498-499
PreciseSQL suite, 498
TSreorg, 498
 application development tools, 495, 546
Delphi, 495-496, 547-548
PowerBuilder, 495, 552
- ReportSmith*, 548-549
SQL Windows, 495, 550-552
 database design tools, 492-493
ERwin/ERX, 493
S-Designer, 493
SQL Coder, 493-494
 monitoring tools, 78
real-time monitors, 79
threshold monitors, 79-80
- THREAD parameter**, 641
- threads**
 defined, 616
 operating systems, Windows NT, 184
see also lightweight processes; processes; traditional processes
- three-tiered system architecture**, 557-558
- threshold monitoring tools**, 79-80
- throughput**, 42
- TIMED_STATISTICS parameter**, 404, 629
- TKPROF program**, 407-409
 output file (listing), 410-414
 parameters, 407-409
- TM (Transaction Monitors)**
 client-side code, 560
 defined, 559-561
 OLTP enhancement, 259-260
 server-side code, 560
 tuning, 561
- Token Ring networks**, 568-569, 617
- tools**
 analysis tools, 496
 defined, 490
Oracle Mission Control, 496-497
Oracle Trace, 497
 summary, 499
third-party tools, 497-499

- application development tools, 494, 534-536
advantages, 534
defined, 490
Developer/2000, 495, 536-544
first-generation tools, 534-535
modern tools, 535
Oracle Power Objects, 544-546
Power Objects, 495
third-party tools, 495-496, 546-552
tuning automatically generated SQL statements, 535-536
- buying performance monitoring tools, 79
- database design tools
defined, 490
Designer/2000, 490-492
third-party tools, 492-494
- monitoring
performance, 75
EXPLAIN PLAN command, 77
Server Manager, 76
SNMP agents, 76
SQL Trace, 76-77
*SQL*DBA Monitor*, 76
- operating systems, 77-78
- third-party monitoring tools, 78-79
real-time monitors, 79
threshold monitors, 79-80
- TPC (Transaction Processing Performance Council)**
ACID tests (Atomicity, Consistency, Isolation, and Durability), 55-58
defined, 52-54
members, 53-54
publishing benchmarks, 57-58
results of benchmarks, 67-69
- Full Disclosure Report*, 56-58
interpreting spreadsheet, 67-69
rules, 55
standard benchmarks, objectives, 53
steering committee, 54
Technical Advisory Board (TAB), 54
TPC-A benchmark, 59-60
TPC-B benchmark, 60-61
TPC-C benchmark, 61-63
TPC-C/S (client/server) benchmark, 66-67
TPC-D benchmark, 63-64
TPC-E benchmark, 64-66
- trace files**, *see SQL Trace*
- tracks (platters)**, 214
- traditional disk drives**
batch processing systems, 271-272
BLOB systems, 328-329
data warehouses, 309-310
decision support systems, 291-292
OLTP systems, 250-252
- traditional processes**, *see processes*
- training services**, 19
- Transaction Monitors (TM)**
client-side code, 560
defined, 559-561
OLTP enhancements, 259-260
server-side code, 560
tuning, 561
- Transaction Processing Performance Council**, *see TPC*
- transaction tables, rollback segments**, 124
- TRANSACTIONS parameter**, 636
- transactions**
benchmarks, TPC-C, 61-62
committed transactions, 13
completing, 12-13
defined, 12-13, 25, 617
discrete transactions, 435-436
distributed transactions, distributed systems, 378
latency, 214
local transactions, distributed systems, 378
OLTP (OnLine Transaction Processing), 65
processing, 395-396
remote transactions, distributed systems, 378
rollback segments, 123-126
user-transaction profile, flowchart, 605
- TRANSACTIONS_PER_ROLLBACK_SEGMENT parameter**, 636
- Transmission Control Protocol/Internet Protocol**, *see TCP/IP*
- triggers**
alerts, 470
creating, 469, 470-471
defined, 25, 469, 617
event logging, 469
generating data, 469
keywords, 470
qualifying statements, 470
summary, 472
syntax, 470
validating data, 469
viewing, 471-472
see also integrity
- Trusted Oracle7 option**
defined, 15
parameters, list of, 642-643
- TShreorg**, 498
- tuning**
goals
connectivity, 43
fault tolerance, 43
load time, 44
response time, 42-43
throughput, 42

memory
buffer cache, 107-108
operating system, 99
private PL/SQL areas, 100
areas, 100
private SQL areas, 100
shared pool, 100-106
methodology, 44
analyzing results, 50
determining solution to problems, 48-49
examining system for problems, 45-46
finding cause or performance problems, 47-48
goal setting for optimal performance, 48-50
performance
clients, 85
networks, 85-86
servers, 85
Tuxedo systems, features, 344
two-phase commit, 617
two-tiered system architecture, 556

U

UNIQUE constraint, 465
unique indexes, 149, 427
units of measurements, conversions, 28
 binary storage units, 29-30
 powers of 10, 29
UNIX, 530-531
 architecture, 192
 development of, 191
 I/O subsystem, 197-198
 asynchronous I/O, 198-200
 file system, 198
 raw device interface, 198
 memory, 192-193, 531
 reducing unnecessary memory usage, 193

SGA tuning, 194-195
user capacity, 195-196
networks, 196, 531
SCO UNIX, 196-197
Solaris, 197
tuning, 575
UnixWare, 197
new features
benchmarking, 200
cache affinity, 202
disabling preemptive scheduling, 201
ISM (Intimate Shared Memory), 201
load balancing, 202
post-wait semaphores, 201
parameters
asynchronous I/O, 198
SHMMAX, 194
SHMSEG, 194
summary of tuning
 guidelines, 202-203
UNIX SVR4, 191
UnixWare
 buffer cache, tuning, 193
 networks, 197
 parameters
 asynchronous I/O, 199
 shared memory, 194-195
updatable snapshots, 374
UPDATE statement
 defined, 617
 execution plan, displaying with EXPLAIN PLAN command, 414-416
updating rollback segments, 128
USE_CONCAT hint, 483-484
USE_MERGE hint, 484
USE_NL hint, 485
user processes (client processes), 10
user-transaction profile, flowchart, 605
USER_CONS_COLUMNS view, 467
USER_CONSTRAINTS view, 467
USER_DUMP_DEST parameter, 405, 637
USER_TRIGGERS view, 471
V
V\$FILESTAT dynamic performance table, disk access information, 597
V\$LATCH dynamic performance table, latch contention data, 601
V\$ROLLSTAT dynamic performance table, dynamic rollback growth data, 600
V\$ROLLSTAT table, rollback segment data, 129
V\$ROWCACHE table, data dictionary statistics, 103
V\$SYSSTAT dynamic performance table, recursive calls data, 599
V\$SYSSTAT table, buffer cache statistics, 107
V\$WAITSTAT dynamic performance table, rollback contention data, 599
V\$WAITSTAT table, rollback segment data, 127
validating data, triggers, 469
VARCHAR2 data type, 510
variables, binding, 401
viewing
 constraints, 467-469
 triggers, 471-472
views
 defined, 6, 617
 dynamic performance tables, 74-75
virtual memory
 defined, 25, 211, 617
 paging, 24, 211
 swapping, 25, 211

W-X-Y-Z

warehouses, *see data warehouses*

WebServer system

- archive logs, 389
- block
 - buffers*, 388
 - size*, 389
- characteristics, 386-387
- checkpoints, 389
- clusters, 389
- contention, latch and rollback, 389
- defined, 18, 368, 386
- design considerations, 387-388
- enhancements, 389-390
- hash clusters, 389
- library cache, 388

multiblock reads, 389

Parallel Query option, 389

Parallel Server option, 389

spin counts, 388

summary, 390

tuning considerations, 388-389

Windows 3.1

- memory, 528
- networks, 528-529

Windows 95

- 32-bit support, 529-530
- memory, 530
- networks, 530
- Oracle support, 530

Windows for Workgroups

3.11

- memory, 528
- networks, 528-529

Windows NT, 183

16-bit applications, 527

architecture, 183-184

defined, 527

I/O performance, 528

I/O subsystem, 186-187

memory, 185, 527

reducing unnecessary

memory usage, 185

SGA tuning, 185-186

user capacity, 186

networks, 186

tuning, 575

summary of tuning

guidelines, 187

threads, 184

words, 29

Workgroup Server, 15-16

write caches, disk arrays, 228-229