



ORACLE[®] DBA SURVIVAL GUIDE

Joseph B. Greene

201 West 103rd Street
Indianapolis, Indiana 46290

I came into this world knowing nothing. Therefore, this book is dedicated to all the people along the way who took the time to teach me something.

COPYRIGHT © 1995 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: 0-672-30681-6

Library of Congress Catalog Card Number: 95-67649

98 97 96 95 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 95-1 shows that the first printing of the book occurred in 1995.

Composed in New Century Schoolbook 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.

PRESIDENT AND PUBLISHER	Richard K. Swadley
ACQUISITIONS MANAGER	Greg Wiegand
DEVELOPMENT MANAGER	Dean Miller
MANAGING EDITOR	Cindy Morrow
MARKETING MANAGER	Gregg Bushyeager

ACQUISITIONS EDITOR
Rosemarie Graham

DEVELOPMENT EDITOR
Todd Bumbalough

**SOFTWARE DEVELOPMENT
SPECIALIST**
Steve Flatt

PRODUCTION EDITOR
Nancy Albright

TECHNICAL REVIEWER
Byron Pearce
Mark Gokman

EDITORIAL COORDINATOR
Bill Whitmer

**TECHNICAL EDIT
COORDINATOR**
Lynette Quinn

FORMATTER
Frank Sinclair

COVER DESIGNER
Tim Amrhein

BOOK DESIGNER
Alyssa Yesh

**PRODUCTION TEAM
SUPERVISOR**
Brad Chinn

PAGE LAYOUT
Louisa Klucznik
Brian-Kent Proffitt
Tina Trettin
Susan Van Ness

PROOFREADING
Nancy Price
Brian-Kent Proffitt
Erich Richter
Susan D. Van Ness
Paul Wilson

INDEXER
Cheryl Dietsch

Overview

Preface	xvii
Introduction	xviii

PART I THE JOB OF THE ORACLE DBA

1 The World of a Database Administrator	3
2 The Database Administrator's Job Description	13
3 History and Development of Databases and Oracle	43
4 Oracle and Its Environments	55
5 The Tools of the Trade	75

PART II UNDERSTANDING HOW ORACLE WORKS

6 How the Oracle RDBMS Works	97
7 Oracle Memory Structures	109
8 Oracle Files	123
9 Oracle Processes	137
10 Basic Oracle Database Objects	153
11 Oracle System Privileges	171
12 Oracle Object Privileges	191
13 Roles and Grants	205
14 Backup and Recovery	223

PART III INSTALLING AND UPGRADING THE ORACLE SOFTWARE

15 The Life Cycle of an Oracle Database	251
16 Choosing Products and the Environment for Your Oracle Database	261
17 Planning an Oracle Installation	277
18 Oracle Installations	295
19 Planning an Oracle Upgrade	311
20 Oracle Upgrades	321

PART IV DEVELOPING A DATABASE ADMINISTRATION SCHEME

21 The Database Administration Scheme	329
22 Laying Out a Database	339
23 A Routine Maintenance Schedule	351
24 Developing Scripts to Automate Tasks	363

PART V THE DAILY ROUTINE	
25 The “Typical” Day	383
26 User Account Maintenance	393
27 Tablespace Maintenance	409
28 Table and Index Maintenance	423
PART VI MONITORING THE DATABASE’S HEALTH	
29 The Health of a Database	447
30 Routine Monitoring	457
31 Auditing	479
32 Tuning the Database	495
33 Looking Toward the Future	517
PART VII DEALING WITH PROBLEMS	
34 When Problems Occur	531
35 Space Problems	547
36 Instance and Application Crashes	563
37 When the Database Is Too Slow	575
38 Troubleshooting Checklist	585
PART VIII SUPPORTING USERS AND DEVELOPERS	
39 Sound Database Object Design	593
40 Query Optimization	611
41 Keeping Current as a DBA	629
PART IX ADVANCED ORACLE TECHNICAL FEATURES	
42 Rollback Segments	641
43 Locks	649
44 Parallel Processing Options	655
45 Packages, Procedures, and Triggers	663
46 Client-Server and Networking	673
47 Where To Next?	681
48 Oracle Workgroup Server and Oracle 7.2	687

APPENDIXES

A	SQL Commands	695
B	Glossary of Terms	701
C	SQL*Plus Features	705
D	SQL*DBA Features	711
E	SQL*Loaders	715
F	Import and Export	719
G	Where to Get More Information	725
H	Sample System Configuration Analyses	727
I	The Disk Contents	731
	Index	737

Contents

Introduction	xviii
PART I THE JOB OF THE ORACLE DBA	1
1 The World of a Database Administrator	3
The Concept of a Database Administrator	7
Variety of Responsibilities	8
Summary	11
2 The Database Administrator's Job Description	13
Types of Database Administrator	15
The Full-Time Database Administrator	18
Developers Acting as Database Administrators	23
Scientists, Engineers, and Other Users as DBAs	28
Database Administrator Tasks	32
Is the Database Secure Enough?	33
Does the Database Perform Well Enough?	35
Is the Data Accurate?	37
Is the Data Stored in a Logical and Accessible Manner?	37
Interfacing with System Administrators and Other Support Staff ..	37
Understanding the Application Needs	39
Summary	42
3 History and Development of Databases and Oracle	43
Computerized Data Storage	44
The First Databases	46
Relational Databases	48
Oracle's History	50
The Current Database Market	51
What Next?	52
Summary	53
4 Oracle and Its Environments	55
What Is Oracle?	56
Alternative Architectures	57
Oracle Development Tools Versus the Database	61
Oracle Utilities	63
Third-Party Products	66
Assembling the Parts into an Architecture	69
Summary	73

5	The Tools of the Trade	75
	The Basics: SQL*DBA	78
	The Next Generation: Oracle Server Manager	81
	Personal Oracle7 for Microsoft Windows	82
	The Command-Line Interface: SQL*Plus	88
	Import and Export	90
	Loading Data from External Systems: SQL*Loader	91
	Third-Party Products	93
	Locally Developed Tools	93
	Summary	93
PART II	UNDERSTANDING HOW ORACLE WORKS	95
6	How the Oracle RDBMS Works	97
	Overview	98
	The Oracle Processes	101
	Memory and Speed	102
	Disk Storage	104
	Multiprocessing and Microsoft Windows Configurations	106
	Summary	107
7	Oracle Memory Structures	109
	Overview	110
	The System Global Area (SGA)	112
	The Program Global Area (PGA)	117
	User Work Spaces	119
	Summary	120
8	Oracle Files	123
	File Locations	125
	Data Files	127
	Redo Log Files	129
	Archive Log Files	131
	Control Files	133
	Initialization Files	133
	Log and Trace Files	135
	Summary	136
9	Oracle Processes	137
	Overview	138
	The System Monitor	142
	The Process Monitor	144
	The Database Writer	144

	The Log Writer	147
	The Archiver	148
	The Recoverer	148
	The Lock Writer	148
	Dedicated Server Processes	149
	Multi-Threaded Server Processes	149
	SQL*Net Listeners	150
	Parallel Query Processes	150
	Oracle 7 for Microsoft Windows	151
	Summary	151
10	Basic Oracle Database Objects	153
	Overview of Storage and Access	154
	Tables	159
	Indexes	162
	Views	164
	Synonyms	166
	Stored Procedures	167
	Clusters	168
	Sequences	169
	Summary	169
11	Oracle System Privileges	171
	Overview of Oracle Privileges	172
	Overview of System Privileges	176
	The User Privileges	178
	The Developer Privileges	178
	The “Any” Privileges	180
	The Database Maintenance Privileges	185
	The Monitoring Privilege	187
	Typical Privilege Sets	188
	The “Any” Privilege Sets	189
	Summary	190
12	Oracle Object Privileges	191
	Overview	192
	Object Privileges	196
	Using Dummy Object Owners	198
	A Typical Privilege Scheme	199
	Summary	203
13	Roles and Grants	205
	Introduction to Grants	206
	Introduction to Roles (Version 7 Feature)	209
	Grants Without Roles	212

	A Typical Privilege Scheme	212
	Use of Scripts to Capture Privilege Sets	220
	Summary	221
14	Backup and Recovery	223
	The Importance of Backups	224
	Overview of Oracle Backup Schemes	225
	Archive Log or No Archive Log	228
	Cold Backups	229
	Warm Backups	236
	Exports	238
	Which Scheme to Choose	239
	Rotating Backup Schemes	242
	Automated Backup Schemes	245
	What About Mirrored Disks?	247
	Summary	248
PART III INSTALLING AND UPGRADING		
	THE ORACLE SOFTWARE	249
15	The Life Cycle of an Oracle Database	251
	Product Selection	254
	Planning the Installation	255
	Installation	256
	When to Upgrade	257
	Planning Upgrades	258
	Upgrading the Oracle Software	259
	Summary	260
16	Choosing Products and the Environment	261
	Getting All the Pieces	262
	Host-Based and Server-Based Architectures	268
	Client-Server Architectures	271
	Dealing with Vendors	274
	Summary	276
17	Planning an Oracle Installation	277
	Starting with Business Needs	279
	The Installation and Configuration Guide	281
	The README File	284
	Oracle's Recommended Layout	285
	Calculating Data Space	288
	Memory Area Planning	289
	Process Planning	289

	Developing Your Own Installation Plan	290
	Some Good Reviewers	292
	Summary	292
18	Oracle Installations	295
	Overview	296
	Starting with a System Backup	298
	The Oracle Installer	299
	Installing the Oracle Application Software	301
	UNIX Installations	303
	Creating a Database with the Installer	304
	Dealing with Installation Problems	305
	Manually Creating a Database	307
	Summary	309
19	Planning an Oracle Upgrade	311
	Overview	312
	Storing the New Software	314
	The Importance of the README File	314
	Changes Needed in the Database	315
	Other Factors to Consider	315
	The Backout Plan	317
	Lining Up Support	318
	Summary	319
20	Oracle Upgrades	321
	The Oracle Installer: To Use or Not?	323
	Another Backup Lecture	324
	Loading and Linking the New Software	324
	Upgrading the Database	325
	When Problems Occur	325
	Summary	326
PART IV DEVELOPING A DATABASE ADMINISTRATION SCHEME		327
21	The Administration Scheme	329
	What Is a DBA Scheme?	330
	Technical Factors to Consider	333
	Matching User Requirements	335
	Preparing for the Future	336
	Summary	338
22	Laying Out a Database	339
	Overview	340
	Data Files	341
	Control Files	344

	Online Redo Log Files	345
	Archive Log Files	346
	The Configuration Process	346
	Expansion of the Database	348
	Summary	349
23	A Routine Maintenance Schedule	351
	Overview	352
	Starting with User and System Processing Schedules	355
	Types of Activities	357
	The Daily Schedule	359
	The Long-Term Schedule	360
	Summary	361
24	Developing Scripts to Automate Tasks	363
	Automation of DBA Tasks	365
	SQL Scripts and System Scripts	367
	Automatic Job Submission Utilities	373
	Developing and Testing Scripts	377
	Monitoring the Results	378
	Summary	379
PART V	THE DAILY ROUTINE	381
25	The “Typical” Day	383
	Scheduled Events	386
	Monitoring	386
	User Support	387
	Problems	388
	If There Is Any Time Left	390
	Summary	391
26	User Account Maintenance	393
	User Maintenance and the Security Scheme	394
	Using System Logon IDs for Access	396
	Adding New Users to the Database	399
	Changing User Access Rights	402
	Deleting Users from the System	406
	Temporarily Disabling Users	407
	Summary	408
27	Tablespace Maintenance	409
	Care and Feeding of Tablespaces	410
	Monitoring and Planning	416
	Typical Problems and Their Solutions	418
	Summary	422

28	Table and Index Maintenance	423
	Care and Feeding of Tables and Indexes	424
	Monitoring Tables and Indexes	434
	Typical Problems and Their Solutions	436
	Fragmentation	442
	Summary	443
PART VI	MONITORING THE DATABASE'S HEALTH	445
29	The Health of a Database	447
	What Is a "Healthy" Database?	448
	Monitoring Programs	451
	Auditing	454
	Tuning	455
	Summary	455
30	Routine Monitoring	457
	A Routine Monitoring Program	458
	Scripts and Reports	460
	Utilization Monitoring	462
	Tuning Monitoring	465
	Security Monitoring	470
	Configuration Monitoring	474
	Third-Party Tools	477
	Summary	478
31	Auditing	479
	Overview	480
	Oracle Auditing Events	483
	Auditing and Performance	491
	Auditing as Part of Security Monitoring	491
	Deciding What to Audit	492
	Summary	494
32	Tuning the Database	495
	What Is Tuning?	496
	What Can You Control?	498
	Host Computer Indicators	504
	Oracle Resource Contention	506
	A Tuning Checklist	513
	Summary	514
33	Looking Toward the Future	517
	Knowing Where the Database Is Going	518
	When More Is Needed	522

Proving Your Case	523
Summary	527
PART VII DEALING WITH PROBLEMS	529
34 When Problems Occur	531
Routine Requests, Problems, and Real Problems	533
Classifying the Problem	536
Resources to Identify the Problems	539
Unknown Problems and Approaches	541
Supporting Resources	542
The Ten Most Common Problems You'll Face	543
Service Level Agreements	544
Summary	545
35 Space Problems	547
Identifying the True Problem	548
Cleaning Out Tablespaces	551
Expanding Tablespaces	553
Compressing the Number of Extents	555
Alternatives: Reducing Data Storage	559
Keeping the Data Definition Language (DDL)	560
Summary	560
36 Instance and Application Crashes	563
Tracing the Problem	564
Log Files Can Help	566
Operating System Conflicts	569
Expanding Oracle Resources for Applications	570
Reducing Oracle Resources for Applications	571
When to Call Oracle	571
Summary	573
37 When the Database Is Too Slow	575
When Is a Database Too Slow?	576
Managing User Expectations	578
Checking Tuning of the Database	579
Application Tuning	580
When Additional Capacity Is Required	581
Summary	583
38 Troubleshooting Checklist	585
Summary	589

PART VIII	SUPPORTING USERS AND DEVELOPERS	591
39	Sound Database Object Design	593
	Overview	594
	Normalization and Table Design	597
	Table Design Modifications for Decision Support	600
	When and How to Use Indexes	601
	Naming Conventions	604
	Summary Tables Versus Views	606
	Sizing Tables	606
	Sizing Indexes	609
	Summary	610
40	Query Optimization	611
	Factors Designers Can Control	613
	The Rule-Based Optimizer	616
	The Cost-Based Optimizer	617
	Execution Plans	619
	Hints	622
	Indexes	624
	General Guidelines	624
	The Value of Experimentation	626
	Summary	626
41	Keeping Current as a DBA	629
	The Changing Environment	631
	Training Vendors	632
	Books	634
	A Test Instance	635
	The Internet	635
	User Groups	637
	Summary	638
PART IX	ADVANCED ORACLE TECHNICAL FEATURES	639
42	Rollback Segments	641
	Introduction	642
	Special Storage Considerations	644
	Common and Confusing Error Messages	645
	Setting Up Rollback Segments	647
	Summary	648
43	Locks	649
	Overview	650
	Types of Locks Applied	651

Freeing Locks	653
Determining When an Application is Waiting for a Lock to Release	653
Summary	654
44 Parallel Processing Options	655
Types of Parallel Processing	657
When to Use Parallel Processing	658
Distributed Databases Versus Parallel Servers	658
Multi-Threaded Servers	659
Asynchronous Database Writers	660
Parallel Query	661
Parallel Recovery	661
Summary	662
45 Packages, Procedures, and Triggers	663
Software Stored in the Database	664
The Object-Oriented World	666
Trigger Types and Uses	667
Database Procedures	668
Database Packages	670
When to Use Packages, Procedures, and Triggers	671
Summary	671
46 Client-Server and Networking	673
What Is Client-Server?	674
Typical Client-Server Architectures	678
Tricks to Administering a Client-Server Database	679
Distributed Databases	680
Summary	680
47 Where To Next?	681
The Oracle Environment	682
The DBA's Job	683
Tips to Make Life Easier	684
The Future	685
48 Oracle Workgroup Server and Oracle 7.2	687
First Impressions of the Workgroup Server Concept	688
Oracle 7.2	690
Rumors and Gossip About Oracle 8	691
Summary	691

APPENDIXES	693
A SQL Commands	695
Object Creation Commands	696
Object Modification Commands	698
Object Deletion Commands	699
B Glossary of Terms	701
C SQL*Plus Features	705
Calling SQL*Plus	706
Output Formatting	706
Working with SQL Files	708
SQL*Plus Versus SQL*DBA	709
D SQL*DBA Features	711
Command Line Versus Screen Mode	712
Menu Interface	712
Monitors	713
Killing User Sessions	713
Oracle Server Manager	713
E SQL*Loaders	715
Using Load Tables and Population Scripts	716
Fixed Column Versus Delimited Formats	716
An Example Control File	717
Useful Optional Parameters	717
F Import and Export	719
Command Line Versus Interactive	720
Import Parameters	720
Requirements for Running Import	721
Oracle Export	721
An Export and Import Example	721
G Where to Get More Information	725
Conventional Addresses	726
FTP Sites	726
World Wide Web Pages	726
Newsgroups	726
H Sample System Configuration Analyses	727
I The Disk Contents	731
Index	737

Preface

Some people lead highly focused careers. I'm not one of those people. Instead, I have tried a number of different jobs, appreciating a variety of assignments and learning new things. This was how I got into Oracle. I was working as a system architect and MIS planner when someone needed to have a database management system. I ordered the system and, because I had worked with Oracle a little before for a test, set up the system for them. Once I set it up, I wound up maintaining the system.

Then a strange thing happened. People saw Oracle DBA work on my résumé and started to ask for my services. Even though I was qualified to perform a number of other tasks and had actually spent more time working in those areas, I received assignment after assignment as an Oracle DBA. Some of these assignments were extremely challenging, working on extremely large databases and integrating a wide variety of client-server products into the environments. Over time, I became very comfortable as an Oracle DBA.

Then I saw a request on the Oracle Internet Newsgroup for authors to write books on Oracle. I had always wanted to write a book, so I jumped at the opportunity. Practical people want to write books to further their careers or get a publication on their resumes. I just wanted to try something new and perhaps spread the knowledge that I had been gaining through my consulting assignments. I had discovered that most of the places where I started up Oracle databases had similar questions and needs that were not addressed by the books provided with Oracle.

Don't get me wrong. I use the books that come with Oracle (electronic and hard copy) all the time. They are usually correct when you have a specific question on a specific technical topic, but often lack an overall feeling for the job of the DBA. They do a good job as an encyclopedia, but do not present the big picture. That is my goal for Oracle DBA Survival Guide.

Writing this book was definitely a challenge. The folks at Sams (especially Rosemarie Graham, the acquisitions editor) were very tolerant of my schedule demands and I appreciate that.

One of the things that I found to be most interesting as an author is the large number of people who work on a book like this. There are a large number of people who work to acquire the books (in my case Rosemarie Graham), develop the book (Todd Bumbalough), perform technical editing (Byron Pearce and Mark Gokman), pull everything together (Nancy Albright), and perform all of the other services such as preparing graphics, typesetting, and so forth. (I don't know these people's names, but thank them anyway).

Finally, there are a large number of people who I would like to thank for contributions that they have made to my development. My Mom brought me into this world

and spurred me along all the way (by the way, Mom, I finished my book before you finished yours). I would like to thank all my bosses and co-workers at Booz Allen & Hamilton for the professional development that they provided me—special thanks to Ed Moore for teaching me to use graphics. Finally, I would like to thank my friends, and especially my wife Vicki, for being understanding about my not being around all those months while I was locked up in the study.

Oracle DBA Survival Guide does not replace the reference books. You still need to look at the Installation and Configuration Guide specific to your operating system and the version of Oracle when performing installations. There are a number of things that change all of the time and you need these specific references. Oracle DBA Survival Guide provides the overall concepts and some of the glue that enables you to know which functions you should be performing and why. Remember, the SQL Language Reference is always available to look up the exact syntax once you know what you are trying to do. I trust this book helps you get your job done more efficiently.

Introduction

Oracle DBA Survival Guide focuses on the job of the database administrator. Much of the text describes Oracle technical topics, but they are presented as background material to understand how to deal with problems and why things work the way they do. Practical experience and solutions to problems that I have encountered are inserted along the way. The book is divided into many topics to enable you to focus on those that are most important to your immediate situation. You can save the remaining chapters for a cold winter night by the fireplace.

This is a thick book. Its usefulness on the job will be determined by how quickly you can find the information that you need. The following quick-access tools have been used:

- ◆ Summary lists of contents at the beginning of each part and each chapter
- ◆ Liberal use of bullet-points and lists
- ◆ Note, Tip, Caution, and Warning boxes
- ◆ Checklists
- ◆ Illustrations, screen shots, and tables for ready reference

Part IV focuses on the development of a database administration scheme. This is a turning point for the book because it marks where the reader has been presented

with the background material, tools, and alternatives needed to perform the job of the DBA. Here is where the focus shifts from the detailed “how do I” to the formation of an overall vision of what is to be accomplished and how. DBAs should form a clear vision of the “big picture” for several reasons:

- ◆ Many DBAs have extremely heavy workloads; an overall view helps you avoid duplicated or wasted effort (if you’re lucky, you may even get home on time occasionally).
- ◆ Because the database is the central point for many different applications and systems, the DBA is often asked questions that relate to the overall flow of information in the business. There are a lot of people who know their niches very well, but few understand the functioning of the entire organization. The DBA is often asked to provide this understanding for corporate data.
- ◆ Oracle is, by its very nature, a complex system. You need to understand exactly what is going on, exactly what is running, and exactly where everything is to be able to solve problems. Often a problem in one place is actually only a symptom of a larger problem in another part of the system.

Very few, if any, readers will read this book cover to cover in one sitting. However, the task descriptions, experiences, technical background topics, and touch of philosophy (that I cannot help interjecting) provide the balance needed by Oracle DBAs in the field.

ORGANIZATION OF THIS BOOK

Oracle DBA Survival Guide is organized into nine parts, each designed to meet a specific need that a reader may have.

PART I: THE JOB OF THE ORACLE DBA

These chapters provide a warmup for the rest of the book. Part 1 begins with some introductory discussions on the job and the tasks associated with it. An introduction to the history of databases is provided for those who are curious about the origin of these systems. An overview of the Oracle family and how it fits into various computer environments is followed by a discussion of the tools of the trade for the database administrator.

PART II: UNDERSTANDING HOW ORACLE WORKS

Although the primary focus of the book is on the job itself, a little theory is required to ensure that the DBA really understands what is happening when the database is operational. Some of the topics are heavily theoretical, such as discussions on

Oracle memory structures. Others are practical, such as the discussions on backups and privileges. All the chapters benefit DBAs by providing a true understanding of what Oracle is and how it works.

PART III: INSTALLING AND UPGRADING THE ORACLE SOFTWARE

What takes an almost insignificant percentage of the total time on the job, but causes the most worry and concern? Installations and upgrades, in many cases. Installation of the software and creation of a new database is often the first task that a DBA is assigned. These tasks need to be performed correctly. The keys stressed in these chapters are the planning processes and checklist development that can make this a relatively painless evolution.

PART IV: DEVELOPING A DATABASE ADMINISTRATION SCHEME

This is the part mentioned previously in which you take the technical foundations and task descriptions explored earlier and form the overall scheme of operation for your database. Topics discussed include laying out where the files will be located, developing a routine maintenance schedule, and developing scripts to automate many of the DBA tasks.

PART V: THE DAILY ROUTINE

These chapters focus on how to perform the tasks that make up the bulk of the day's work for an Oracle DBA. This includes user account maintenance, tablespace maintenance, and database object maintenance.

PART VI: MONITORING THE DATABASE'S HEALTH

Databases can be very busy places. Data flows in and out. Developers are busily writing new applications. It is important to keep an eye on things within the database so that you can detect situations that may lead to problems before they actually become problems. This part describes some monitoring tasks that you can perform to detect the common problems that grow within a database, such as running out of space, security violations, and a lack of certain key system resources.

PART VII: DEALING WITH PROBLEMS

In the ideal world, everything works perfectly. This is not the ideal world. However, a sound approach to dealing with problems—classifying, determining the cause,

forming solutions, and knowing where to get help—can reduce the number of gray hairs received in the line of duty. This part provides some techniques that can help DBAs deal with problems when they arise.

PART VIII: SUPPORTING USERS AND DEVELOPERS

In many database installations, DBAs are seen as the gurus of data storage. They need to be able to provide guidance to developers and users about how to get the most from their systems. These chapters discuss some of the common subjects of discussion.

PART IX: ADVANCED ORACLE TECHNICAL FEATURES

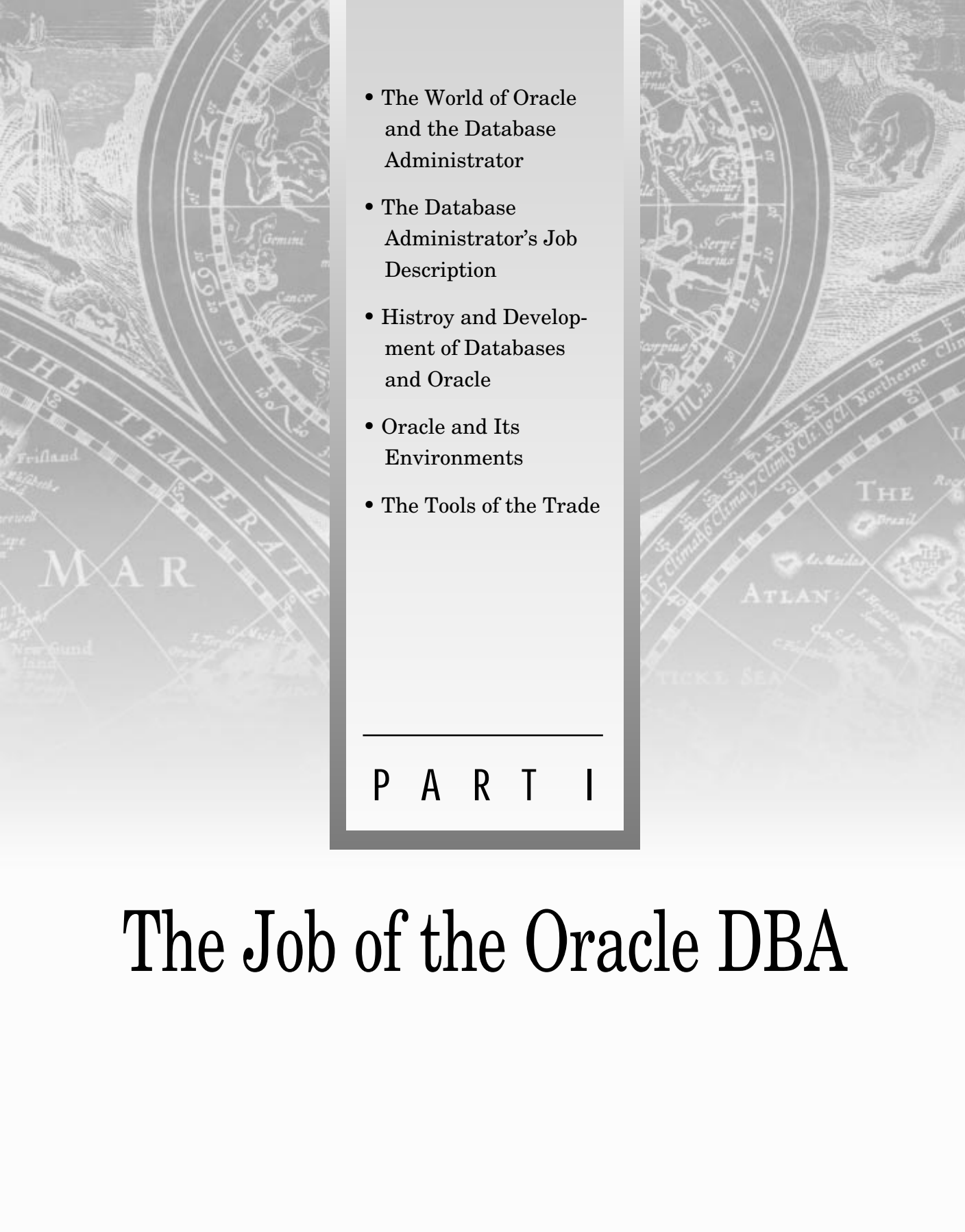
This part covers more detailed technical topics, such as locks and parallel processing, that are not for everyone. You can see these chapters as a place to go with unusual questions or a series of topics that will lift you one level higher in the DBA plane of existence.

APPENDIXES

The appendixes contain command syntax, definitions, and useful lists that will make the job of finding key data a little easier. Almost all this information exists somewhere within the Oracle documentation set. However, these appendixes reduce the number of books you have to look through for answers.

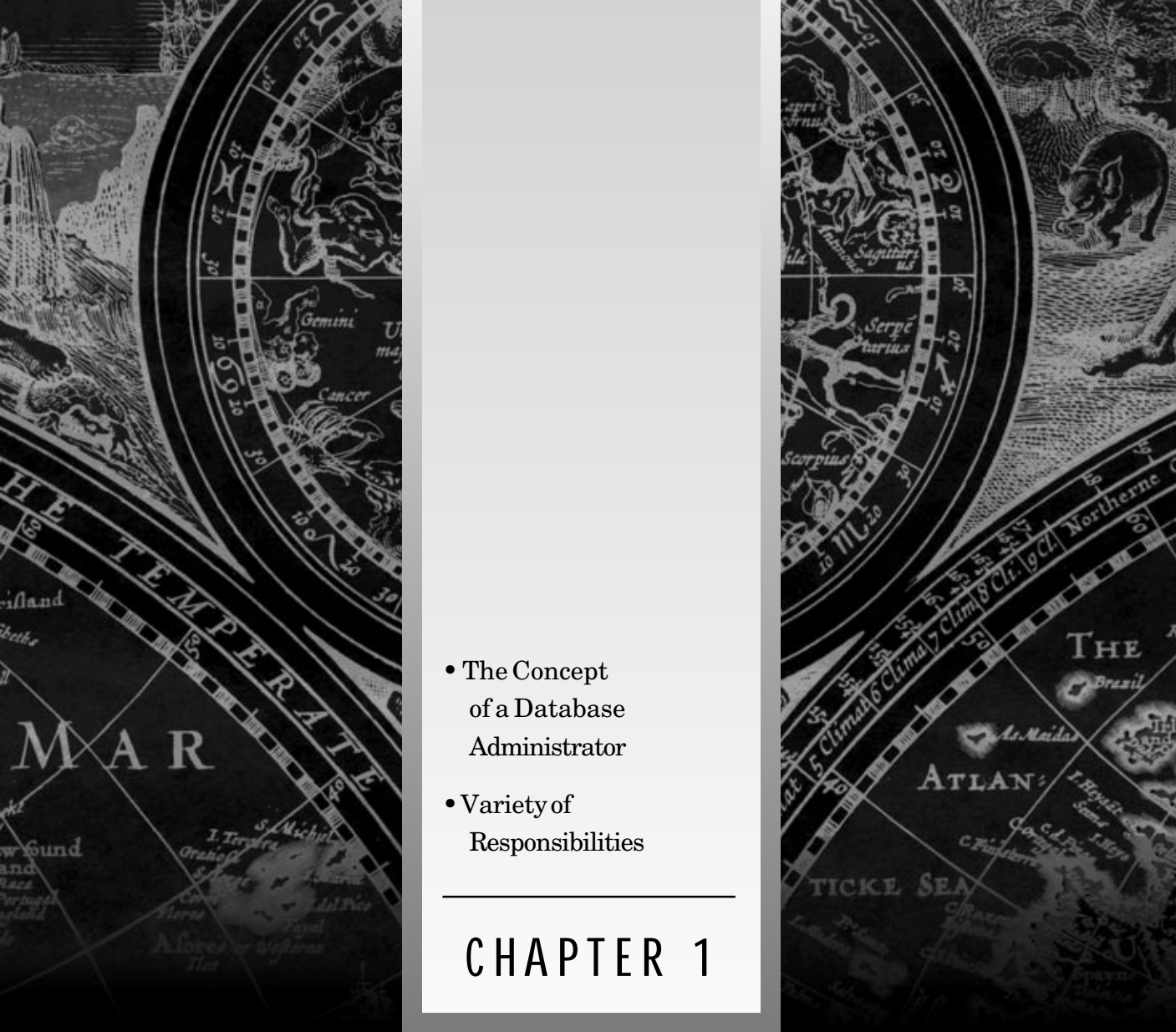
SUMMARY

Remember, you are not expected to read Oracle DBA Survival Guide in one sitting and become an immediate DBA. Do not be intimidated by the large volume of material. Most DBA tasks require information from only one or two chapters at a time. You can learn about the job gradually, as you need it.

- 
- The background of the slide is a composite of two historical illustrations. On the left, a portion of a map is visible, showing a coastline with labels like 'Frissland', 'Cape', 'New Gunt land', and 'I. Terceira'. A large, stylized letter 'M' is prominent. In the center, a circular zodiac wheel is partially visible, with labels for 'Geminus', 'Cancer', and 'Serpens'. On the right, another portion of a map is visible, showing a coastline with labels like 'Brazil', 'I. Madeira', 'I. Hispania', and 'I. Laysan'. A large, stylized letter 'A' is prominent. The overall theme is historical exploration and navigation.
- The World of Oracle and the Database Administrator
 - The Database Administrator's Job Description
 - History and Development of Databases and Oracle
 - Oracle and Its Environments
 - The Tools of the Trade

PART I

The Job of the Oracle DBA

- 
- The Concept of a Database Administrator
 - Variety of Responsibilities
- ## CHAPTER 1

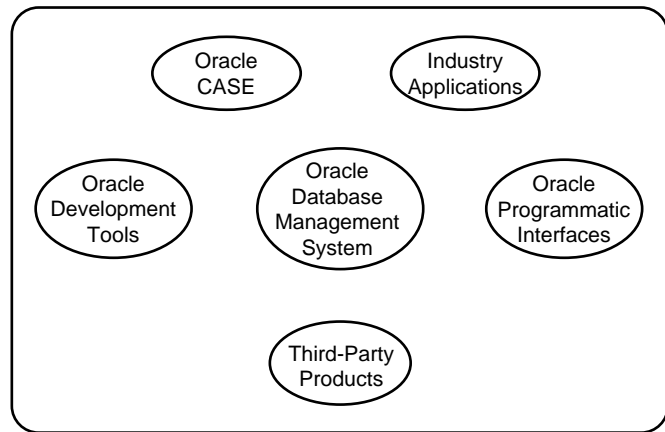
CHAPTER 1

The World of Oracle and the Database Administrator

Vendors often describe their “world”—in the case of Oracle, the term “world” seems to be correct. What started with a relational database management engine has grown into a number of development tools, networking products, computer aided software engineering (CASE) tools, and some complete, industry-specific packages such as Oracle Financials. Part of the challenge faced by Oracle database administrators (DBAs) is to determine what parts of this Oracle world they must support and prepare themselves accordingly. And so begins the challenge.

With many complex topics, it is easiest to start with a brief overview that breaks the subject down into smaller parts and then explore each of these components individually. Figure 1.1 graphically portrays an overview of the Oracle world and some of the interfaces to it that are commonly found.

Figure 1.1.
The world of Oracle.



The Oracle relational database management system (RDBMS) was the initial product offering and is at the center of most Oracle product installations. This product is designed to provide the facilities to store and retrieve information in a variety of formats. The data entrusted to Oracle is stored in a series of data files on disk drives of the computer that is running the Oracle RDBMS. If the RDBMS were limited to a set of disk files, it would be similar to dBASE on a PC. To work in large, multi-user environments, Oracle has had to add a number of items to the RDBMS architecture to provide needed services. First, to increase speed on these large systems and ensure consistency of the data, Oracle uses large areas of computer memory to store data, transactions, and control information. Next, it implements a series of background processes to perform part of the work of storing and retrieving information. This also helps to reduce the complexity of each individual process and increase the speed of processing, especially on computers with multiple CPUs. To support data integrity, Oracle keeps a separate record of transactions that are made

so that the data can be removed or rolled back if a transaction is canceled or the system crashes. In addition, there are several types of logs that record transactions made with the purpose of enabling the data stored in the system to be recovered up to the point of failure even if a disk drive is completely destroyed. Finally, there are a set of utilities that provide DBAs with an interface to the database engine and enables them to perform common tasks such as system startup, data backup and data loading. It can be quite a complex system, but this is the secret of its power. Only a few DBAs have to be concerned with all the technical details of the product, and this knowledge can be built up over time.

Oracle's next major product line is the series of tools that it provides to build applications that interface with the RDBMS. One of Oracle's early product directions was to build a series of fourth-generation language (4GL) and graphical user interface (GUI) tools to enable users to rapidly build applications that interfaced with the Oracle RDBMS. Some of you might be wondering what those terms really mean. Quite simply, Oracle built tools initially that used very high-level (close to the English language) commands to access the database and build applications—the 4GL tools. The GUI (often pronounced gooey) part of the story started with form- and report-building tools that used Lotus-like menus and screen painters to build applications. Now, some purists may contend that this did not constitute a pure GUI. Well, one of the main themes in this book is not to dwell on theories and academic debates. For real-world purposes, this definition should be close enough to the truth, if there is such a thing. Anyway, Oracle has continued with products that use pure GUI interfaces, such as those found in Microsoft Windows, the Macintosh operating system, and the Motif environment. In parallel with this, they developed interfaces (Oracle pre-compilers) that can be used to access an Oracle database from within a program written in a common programming language, such as C or COBOL. The key point to understand from this discussion is a product direction of using a mix of high-level tools and programmatic interfaces to build applications. Many competing database vendors designed their application development tools only around traditional programming languages such as C, COBOL, etc. Others designed their development tools only around the 4GL approach. Oracle has a tendency to support a wide range of tools, which has probably helped their market share because each type of development tool is suited to a different type of development effort.

A third product line that has been around for several years is a set of computer-aided software engineering (or CASE) tools. These products are designed to enable you to capture, in a graphical format, a representation of the application that you wish to build for review and analysis. There are a number of information engineering rules and analyses that can be applied to the design to make it more sound. If desired, you can even instruct the computer to generate major pieces of the application automatically.

Another major product line currently being marketed is a series of applications developed for certain industries. This is designed to satisfy those organizations that wish to purchase turnkey solutions from a single vendor. These applications provide most of the common functions needed in such industries as financials (accounting, inventory, and so forth) and manufacturing (production scheduling and tracking). They, of course, are based around tables within an Oracle database, but they also consist of a fairly complex array of programs that would take an organization some time to develop on its own.

An important set of products in modern, multi-vendor, and client-server environments includes the networking products provided by Oracle and other vendors. These products allow developers to write applications on one platform (for example, a PC) that access data located in an Oracle database on another type of computer (for example, a UNIX server). It also allows different Oracle databases to exchange information, both on an as-requested basis and through automatic synchronization of specified data tables (the distributed database option). Finally, there are Oracle and third-party gateways that allow Oracle databases to communicate with other vendor's databases, such as those from Sybase or IBM's DB2.

The final component in the world of Oracle comprises the interfaces provided to products that are not produced by Oracle Corporation. In many installations, this is a major part of the environment that the DBA has to understand and support. Companies such as Powersoft have made significant inroads with sophisticated yet simple application development tools that have become standards in many organizations. Therefore, it is important for a DBA to understand how these tools interface to the database.

This is the world of Oracle as it exists today. It is interesting to note the directions in which Oracle seems to be heading. We can only guess these directions based on company statements and some applied logic, but here is my best guess. Oracle has placed a strong emphasis on getting into the multimedia environment. It seems to envision an Oracle data storage utility as the basis for distributing multimedia services, such as on-demand television programs. Another direction seems to be acquiring and/or developing tools to compete in the object-oriented development tool environment with Powerbuilder. A final observation is that Oracle, and most other RDBMS vendors, seem to be moving towards the capability of storing objects (blobs of any type of data such as X-ray images, songs, or videos) within their database. Although many smaller vendors have designed pure object-oriented databases, Oracle and other major RDBMS vendors seem to be migrating to object orientation gradually.

THE CONCEPT OF A DATABASE ADMINISTRATOR

Before starting work as an Oracle database administrator, it is interesting to consider how this job came into being. Early data storage mechanisms, such as tape, card decks, and flat files were owned and maintained by the programmers who developed the application (only they understood how these beasts were put together). Organizations tired of writing search and maintenance routines for each new application, and data set sizes grew, especially those stored online on magnetic disks. Specialized software packages designed to manage stored data were developed. Over time, the complexity and utilities for these database management systems grew. This created the need for a database administrator who understood the inner workings of these packages and could get the most out of them. It also created careers for many individuals.

It is sometimes interesting as a consultant to guess the types of computers that an organization has had in the past based solely on the job titles. Personnel systems tend to change very slowly and the job responsibilities may change, but the titles remain. IBM actually published a fair amount of literature to its mainframe customers regarding recommended organizations for the data center. These organizations tend to have groups such as operations support, technical support, database administration, and operations. All of these are headed by a separate manager reporting directly to the data center manager (who often goes by the title of director or chief information officer). Organizations that grew up with VAXes and other minicomputers tend to have groups of operators (perhaps with a shift supervisor, but not often), an all-powerful system administrator (or one per system if you were lucky), and the database administrator. This is a warning that you may inherit a legacy and expectations based on what IMS, DB2, or RdB database administrators did on another computer platform. One of the goals of this book is to provide you with some suggestions as to tasks that the Oracle DBA should be performing, along with the justifications for these tasks.

It is important to remember that you are becoming a specialist in the Oracle relational database management system. Having worked on Sybase, Informix, dBASE, and other database management systems, I appreciate the differences between the various systems. Oracle's sound market share and trends towards down-sizing, right-sizing, client-server, and so forth should give you a warm fuzzy feeling about future job security for talented Oracle DBAs. However, never lose sight of your position in the marketplace. My brother spent a brief tour of duty as a

recruiter in the computer industry. He related to me that one of the hardest parts of that job was when someone with a family to support came to him after being laid off. The story usually went that they loved his work, felt his \$80,000 salary was justified, but they had to convert from the old Unisys (or Wang or ...) system and the database on which he was an expert did not run on the new platform. In one case, my brother found that there were only two or three installations of that database package left in the entire country. Enough said—if you plan on making a living as a database administrator, it is always wise to keep an eye to what is becoming marketable in the industry.

VARIETY OF RESPONSIBILITIES

In the era of the large, corporate data center, the job descriptions were based on recommended structures from the various vendors. This usually included large staffs full of managers, directors, shift supervisors, and other wonderful titles. Many organizations moved toward departmental computer organizations and the like. This created a number of jack-of-all-trades positions where someone may have been the operator, the database administrator, the system administrator, the manager, and the janitor. Now compound this with the trend toward inexpensive UNIX-based computers, where individuals may act as their own computer staffs for their workstation or a small server. This leads to a world where people ranging from full-time, highly-trained database professionals to mechanical engineers, who are too junior to get out of the assignment, serve as database administrators. This book deals with this wide range of needs and provides help for all those brave souls who call themselves a DBA.

Perhaps it would be useful at this point to discuss how some organizations derive the responsibilities for the database administrator. This could be useful to understand if you are given the opportunity to have input into the discussions of a realistic set of responsibilities for your given circumstances. Recall that the IBM and DEC literature guided the establishment of job descriptions in many data centers in the past. The following list of factors that help determine the DBA job description results from observations during consulting:

- ♦ No one else will learn new technologies, so if it is new, that person gets it.
- ♦ We can afford only one person to work on the UNIX pilot project that we are conducting, so that person will be DBA, system administrator, and operator.
- ♦ It is your research project and we cannot afford to hire a dedicated computer staff for you.

- ◆ We went to the Oracle management seminars and this is what they told us.
- ◆ We brought in some consultants who said they knew everything and this is what they told us.
- ◆ We can't let the data center get their hooks in us again, so one of us has to do it all.
- ◆ Tech support owns the box and they will let us do only certain things.
- ◆ Our director got the data center director mad, so they will not help us.
- ◆ Long backups are a pain; no one else wants to do them, so we assign it to the DBA.

Although I said that there is no one set of duties for a database administrator, I'm going to take a shot at a list of such duties anyway. Consider this list as food for thought:

- ◆ Perform data backups and recoveries (a favorite of mine, so I get to list it first). Depending on your local computer culture, the actual backups may be performed by computer operations or other support staff (especially in large data centers). However, regardless of who is issuing the backup commands, the DBA had better be certain that the backups being performed will support recovery in the event of a loss of data.
- ◆ Install and upgrade the Oracle application software.
- ◆ Start and stop the Oracle instance.
- ◆ Control user access to resources and information.
- ◆ Monitor and allocate storage space for data.
- ◆ Audit database usage.
- ◆ Tune the database.
- ◆ Perform the physical database design (that is, determine which data items are stored in which file and on which disk drive).
- ◆ Be the focal point when problems arise with the database or applications that access the database.
- ◆ Answer user questions.
- ◆ Assist in the development of sound database queries and applications that work with (not against) the database management system.
- ◆ Keep current with database, operating system, and application development technologies.

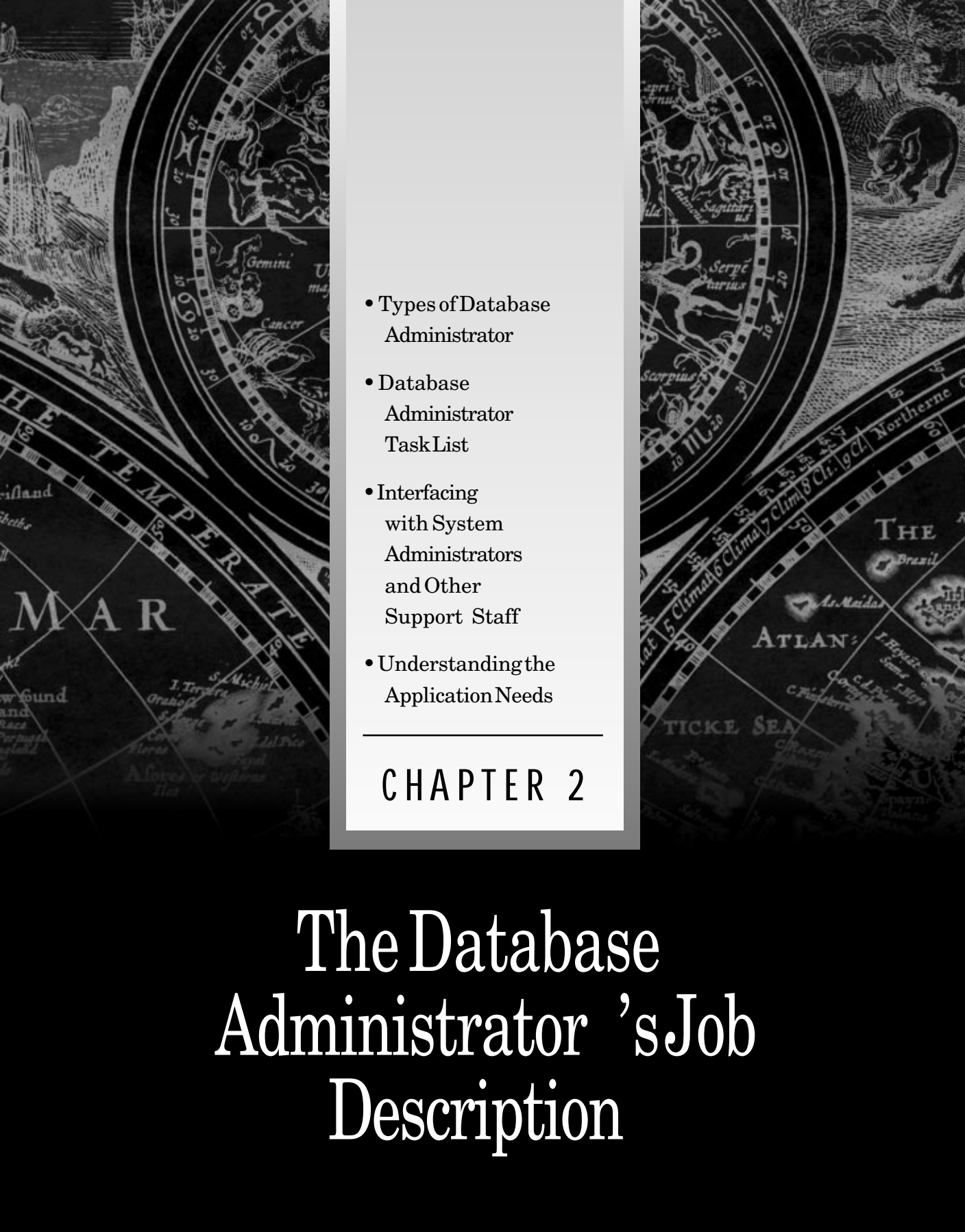
Where do you fit within the realm of possibilities? Consider some of these types:

- ◆ Large UNIX System Database Administrator
 - ◆ Not allowed to touch the operating system, network, or anything other than the database itself (yes, this is a mainframe shop)
 - ◆ Responsible for backing up the data files only
 - ◆ Answers any and all questions about the database
 - ◆ Upgrades the Oracle software, as needed
 - ◆ Develops scripts to help automate many of the database administrative processes
 - ◆ Assists developers in designing tables and queries for high efficiency
 - ◆ Monitors usage, security, and tuning closely
- ◆ Small UNIX System Database Administrator
 - ◆ Serves as backup UNIX system administrator (and he is the backup database administrator)
 - ◆ Develops scripts that back up the entire computer system
 - ◆ Serves as a contact with Oracle when problems arise (he is not expected to have the background to be able to answer difficult questions)
 - ◆ Installs the Oracle software and several off-the-shelf applications packages
 - ◆ Assists developers interfacing to this database with client-server tools
- ◆ Development Group Database Administrator
 - ◆ Serves as UNIX system administrator, database administrator, guru in the C programming language, and software developer
 - ◆ Backs up the computer system
 - ◆ Answers any and all questions other developers may think up
- ◆ Engineering Work Group Database Administrator
 - ◆ Leaves VAX system administration to another contractor
 - ◆ Answers all questions from management and technical staff regarding the Oracle database and all development tools
 - ◆ Backs up the database data files
 - ◆ Develops new engineering applications using Oracle as the data repository

With all that said, remember that the exact tasks you are asked to perform may vary, but it is important to have agreement by all parties what those tasks are and the level of service that you will be providing. If you have other responsibilities that take priority, make sure everyone knows that before problems come up.

SUMMARY

This chapter was a quick philosophical overview of where this book is headed. It stressed the book's philosophy that it should focus on the Oracle DBA job itself. The Oracle relational database management system (RDBMS) was the initial product offering and is at the center of most Oracle product installations. It also was the first of many discussions in this book where you will explore a variety of options as opposed to one simple path. With the wide range of hardware platforms, versions, and user environments, it is impossible to lay out one simple "cookbook" approach to Oracle database administration. Instead, you will learn the key principles of being a DBA and working with Oracle that should allow you to find the correct path in your own particular world. Now, onto the rest of the book!

- 
- Types of Database Administrator
 - Database Administrator Task List
 - Interfacing with System Administrators and Other Support Staff
 - Understanding the Application Needs
-

CHAPTER 2

The Database Administrator's Job Description

This chapter challenges you to consciously consider the job of the database administrator and the possibilities of which tasks are performed by the DBA and which are left to others. All too often, this division of responsibilities just happens. Perhaps something goes wrong, so the most competent individual in the organization is assigned to ensure that it does not happen again. Perhaps it has “always” been done that way. Take some time when reading this chapter to consider which ones should be done by the DBA, given your individual situation.

First, the chapter presents some typical situations that I have encountered in my consulting travels. This material serves to present how others have arranged their job descriptions. Then you will explore a list of tasks and what is involved with these each. Finally, you will examine common threads that should be woven into all database administration schemes. It is important to remember that there is no one absolutely right solution. Instead, you have to find the solution that fits your situation at a given moment. Odds are that as technology progresses, the list of tasks will change, but that is no reason not to have a plan for today.

Recall that in the days when ships were wood and men were iron, data was stored in flat files. There were no database administrators, although there were often senior programmers who kept track of the data stored on various files and tried to set standards for how and where information was stored. Data was often stored in enormous decks of computer punch cards that were stored in large boxes. Many potential computer lovers may have turned away from the field after their deck of punch cards scattered onto the floor just as they were ready to run that job one final time.

Once corporations got hooked on the idea of storing valuable corporate data on computers, money became available to store data on disk drives. At that time, disk drives with under 100M capacity sold for tens of thousands of dollars. This created the beginnings of the database administrator job. People were assigned to ensure that data was not being duplicated, that it was stored in an efficient manner, and that unnecessary data was purged.

Corporate data storage became a full-fledged addiction. Executives loved being able to find information that had been locked away in paper files and to calculate costs and delivery schedules far faster than ever before. In walked software vendors to feed this habit. They noticed that every data application had a large amount of common code. These routines included such basic functions as finding and retrieving a row efficiently and writing a new record. They developed common routines in packages that would be purchased, thereby improving development productivity.

Organization responded well to these packages of data handling utilities that became database management systems. There were a number of different thoughts on how to organize the data (hierarchical, relational, and others), and systems were developed that implemented these different philosophies on different host computer platforms. As these little jewels became more complex, it became a full-time (and fairly well-paying) job to manage this treasure chest of information. The duty of the database administrator was born. In most cases, the DBA was a full-time computer professional who devoted most of his time to maintaining the database.

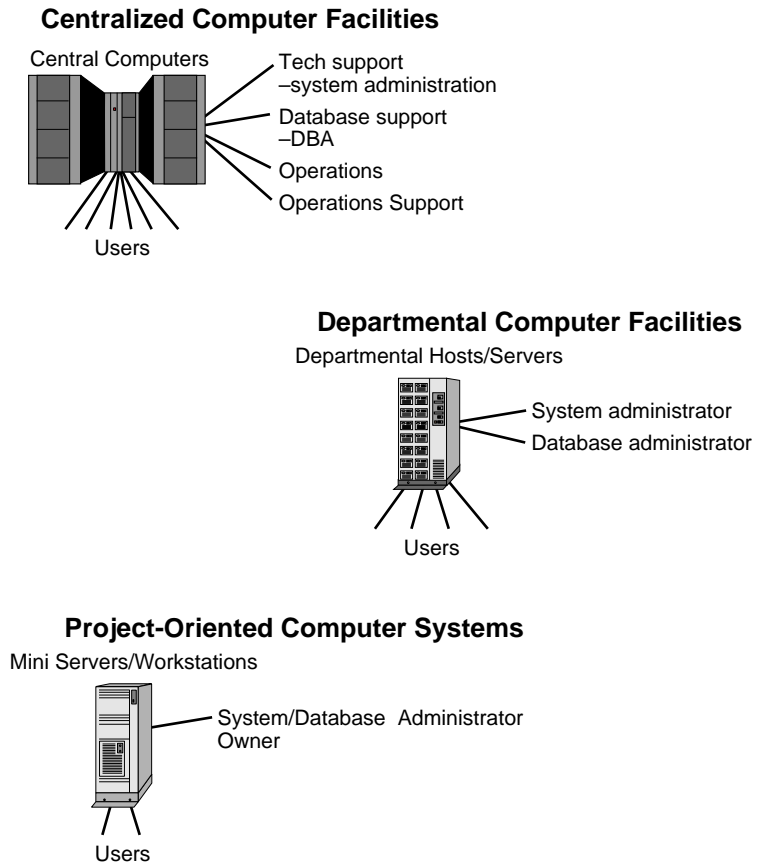
Then came departmental computing. Various business organizations decided that the central data centers were not responsive enough and they implemented their own computer complexes, usually on minicomputers such as the DEC VAX. Vendors built many good database management systems for these platforms, and they needed a DBA. However, in many of these installations, the DBA had other responsibilities, such as being the system administrator or handling some other role within the smaller, departmental computer organization.

Today, we have a mixed bag of database management systems, platforms, and organizational structures (see Figure 2.1). The rise in the number of small (easily fits on a desktop) UNIX computers that have substantial processing capacity enables many small organizations to set up databases on multi-user computers that were previously locked on one person's PC running dBASE. This also usually means that the person who is the dBASE developer is often asked to become a DBA on this multi-user system. Of course, many DBAs still work in large central data centers that are implementing Oracle on mainframes or larger UNIX servers. The departmental folks are also swapping their minicomputers for smaller servers that do more. However, it is often easier to change the hardware and software environment than it is to alter the organizational structure. Try to keep an open mind while reading the next couple of sections. Envision how you think it should be done in your environment. Make notes and try to effect changes when the opportunities present themselves.

TYPES OF DATABASE ADMINISTRATOR

It would be nice if the simple categorization structure described previously covered all possible DBA job descriptions. However, there are many nuances in the roles and responsibilities that should be considered in light of your talents and the "culture" found in your organization. If nothing else, studying other people's work environments is an interesting pastime. When you are a consultant, you need something to get you through certain routine assignments.

Figure 2.1.
Sample DBA environments.



So what variables differentiate the various DBAs out there in the field? The needs of the individual business units drive the DBA job description. There are probably as many variables as there are organizations, but for the most part, I have found the following dominate:

- ♦ Size of system maintained
- ♦ Full-time or part-time DBA status
- ♦ Level of overall system involvement
- ♦ Level of development knowledge required
- ♦ Level of technical database expertise required
- ♦ Corporate politics
- ♦ Specialization

The first factor, size of the system maintained, is one of the most interesting when dealing with Oracle systems. Oracle has developed versions of its software to run on everything from stand-alone PCs to mainframes. Reading forums such as the Oracle newsgroups on the Internet, one gets an appreciation for the fact that different environments face different problems. The larger systems tend to have full-time DBAs who worry more about tuning and disk space management. The smaller installations tend to worry about system capacity and have part-time DBAs. Now that there is a personal version of Oracle running under Microsoft Windows, there are some people who act like dBASE developers and also act as DBA (sort of) for their personal Oracle instance.

The next factor to consider is the percentage of time available to you to perform the DBA tasks. Although this is often determined by the size of the database (specifically the number of tables and users versus the disk storage size), other factors such as the amount of changes, technical complexity, or amount of support required by the user community may shape the decision. The key to remember is to keep the percentage of your time available for the DBA job consistent with the tasks that you agree to perform. No one wins if you (or your manageability) commit to and cannot perform critical tasks.

Another variable that you need to have agreement on when committing to perform database tasks is the level of expertise on the database package that will be expected of you. Many installations purchase commercial applications based on the Oracle database and hire consultants to plan and install the system. The in-house DBA is merely responsible for performing backups, administering user accounts, and monitoring utilization. At other installations, DBAs are expected to do all the design, planning, and implementation. They are often expected to be a fountain of wisdom on all subjects related to Oracle. The trick here is to ensure that your training and access to Oracle resources matches the expectations of you. It is too easy for your company to send you to minimal training and give you little opportunity to keep up with technical topics via seminars, yet expect you to know everything. It is also important to emphasize the continuing education function because the technology continues to change at a rapid pace.

A topic related to database expertise that needs to be defined is the level of support the DBA will provide for developers. Most of the routine tasks performed by the DBA can be planned out and, to a fair degree, automated through the use of scripts and other tools. The greatest challenges that I have found involved supporting developers who are tuning to make that query 20 percent faster or use stored procedures for the first time. Also, if your testing program is good, it will usually be the developers who find problems with the security scheme or bugs with Oracle software. One of these problems can take a few minutes or several weeks to solve. Therefore, if you

are going to be asked to support developers, ensure that you are not booked 100 percent on the routine tasks and that you are provided with resources, such as support contracts, to get help when needed.

Another decision that an organization has to make when defining the DBA job description is the amount of computer system support that the DBA has to provide. In typical mainframe shops, there is a cast of other supporting players to provide support for system administration, security administration, and hardware support. In smaller UNIX shops, all computer-related work may be performed by a single individual who is also the Novell administrator and PC guru. Ensure that adequate time has been factored into the schedule for these functions. If you are new to a particular computer environment, ask someone who is performing a similar job or your vendors to get some estimates on support time.

Finally, it is interesting to review the specialization requirements placed on some DBAs. I worked with one fellow on a very large Oracle data warehouse who spent months working on nothing but application and query optimization. He knew so many little tricks about setting the order of the tables in a query that he could turn 20-minute queries into 5-second performers. This specialty took a long time to develop. If your organization expects such specialization from you, ensure that you are provided the time and training to do the job.

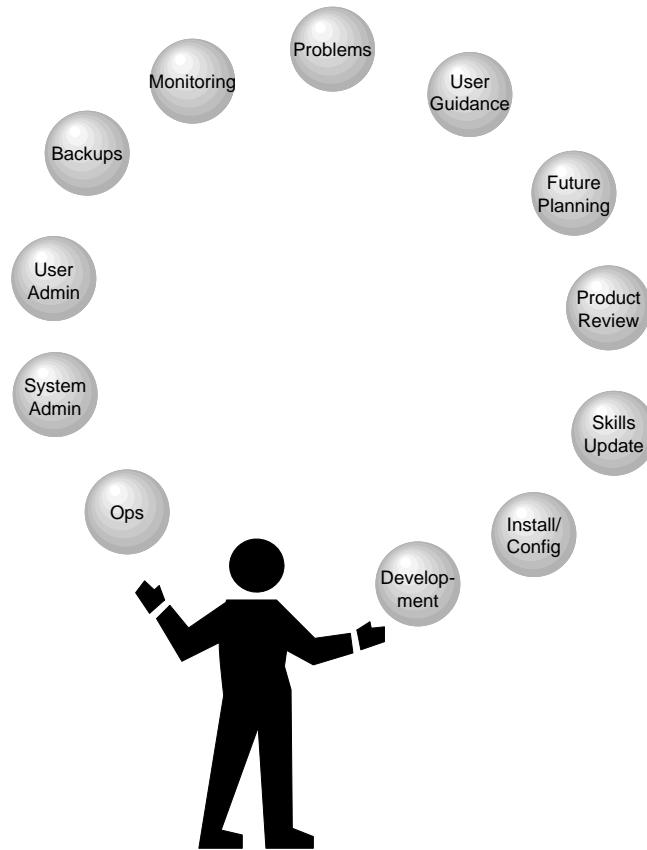
THE FULL-TIME DATABASE ADMINISTRATOR

Perhaps the easiest type of DBA job to describe is that of the full-time DBA. Obviously, there will be a wide variation in the exact job description driven by the needs of the business unit. However, here's a "typical" full-time DBA description to give you a feel for the job.

The full-time DBA is usually involved with larger, more complex systems that require a higher amount of care and feeding. These shops also tend to depend less on outsiders to install and plan their system needs. Finally, because they tend to come much closer to the capacity limits of the computer systems and tools, they tend to find more problems with their systems and support tools such as database management systems. These DBAs need to juggle a number of factors that are pulling on them to try and get what is needed done (see Figure 2.2).

One of the first things that needs to be hammered out is whether the full-time DBA is responsible for operational systems, development support or both. It is important if you support both types of systems to have a firm agreement as to the relative priorities of the two. In most shops, operational problems take precedence over everything imaginable. It is usually the routine work that takes some juggling to balance.

Figure 2.2.
Tasks that the full-time DBA needs to juggle.



Warning

Any system large enough for a full-time DBA that has both development and production responsibilities should have separate database instances for these functions. There are two reasons for this. First, developers can find bugs with the DBMS or operating system that could cause the Oracle processes to crash. Even if they do not crash the system, they can accidentally overwrite data or fill up logs or tablespaces that can cause production activities to stop.

Another important consideration for the full-time database administrator is the division of labor regarding system administration. Some shops may consider that administration of the entire system is part of the DBA's job. This may not be all bad, if the system is small enough or static enough to be managed by a single person. The

DBA is usually the person best suited to lay out the disks, processes, and memory in a way that optimizes the database. Many Oracle functions such as loading software require access to system administrative privileges.

Balanced against the advantages of having all the power (system and database) are the limitations of hours in the day and capability of learning both the operating system and the database management system. An unfortunate problem encountered in a lot of shops that are down-sizing is that database and system administrators are often asked to lay out and install a system after having only a minimal amount of vendor training and little practical experience. A high percentage of the new UNIX/Oracle shops that I have visited wind up performing significant reconfigurations of their systems and databases within a year of initial installation. (This book is a partial solution to this problem.) However, acknowledge the challenge of performing this configuration work and make sure your bosses understand it. If they absolutely need it right the first time, get some outside help from experienced hands who have been there before.

One setup that is useful in smaller shops just large enough for a full-time database administrator is to have the system administrator be the backup database administrator and the database administrator be the backup system administrator. First, no one is invulnerable and the chances of a problem occurring with the system on the day that you are sick are very high according to Murphy's Law. Most useful about this situation is that it enables each person to focus first on their area of responsibility. They set their own areas up properly but work together to gain an appreciation of the needs and patterns of each other. It is useful for shops undergoing a major conversion in the operating system and DBMS when there is not enough time to learn both jobs before startup. It also is sometimes easier to learn from a co-worker who is across the hall in 10-minute sessions rather than by sitting in a classroom for a solid week.

This leads me into another task for the full-time DBA—installation and configuration planning. Most of the managers that I have run across figure that if you are full-time, there is little need to hire outsiders to plan the installation. This can actually be to your advantage (you get things done your way) as long as you are not asked to set up a database and computer system with which you are not familiar. This can be one of the more complex tasks performed by the DBA and requires a sound understanding of both what the database management system needs to work efficiently and what the user applications are going to be doing (is it input/output-intensive, what will the total storage capacity be in one year, and so forth). Do everything you can to ensure that you have been given the proper background (classes, this book) and the time to prepare a sound plan for your system. If you are not given the time and background, relax and do the best you can. You would not be the first Oracle DBA placed in this position (you probably would not even make the top 1,000).

There is a later section of this chapter dealing with each of the tasks that you might be asked to perform. Many of the tasks, such as backup and recovery, also have entire chapters later on in this book. The following is a quick list of the other tasks that you need to factor into your daily schedule:

- ◆ *Backup and recovery.* Even if you have a system administrator, you are the expert on Oracle. You know where to place files, how long archive logs need to be kept, and so forth.
- ◆ *User administration.* This involves adding users to the system, creating roles, granting privileges to database objects to these users and roles, and similar tasks.
- ◆ *Monitoring the health of the database.* The resources of the database need to be monitored to detect problems before they impact users. Typical items to monitor include disk space utilization, system memory resources, tunable parameters, security privileges, and usage of the database.
- ◆ *Dealing with problems.* This is not a perfect world. Problems occur due to bugs in the Oracle software, bugs in the operating system, bugs in your user or purchased software, hardware failures...and the list goes on. The only weapons that can help solve problems are knowledge (Oracle, operating system, your applications, and so forth) and support services (either within your organization, from the vendors, or from third parties).
- ◆ *Providing guidance to users.* Do not be surprised if developers ask you about the most efficient type of loop for a particular circumstance. Users may also come to you with questions about minimizing windows. Finally, unless you are a truly lucky soul, you can count on the “why doesn’t the database run any faster?” questions from your users. Knowledge and experience are needed to provide this type of support. While coming up to speed—and to answer the harder questions later on—it is beneficial to have access to support resources who may have more experience than you.
- ◆ *Future year planning.* Some say that the first casualty in war is the truth. I have observed that the first casualty in a busy computer environment is future planning. Many shops are forced to run from fire drill to fire drill. The unfortunate thing is that a lack of planning tends to promote future crises. It is important to do what you can to ensure that you have time to provide future planning for disk space, processing capacity, software upgrades, and so forth so that you can break out of this cycle of crisis. A useful technique is to bring in a contractor to take over some of the workload for a month or two to help free you up.
- ◆ *Reviewing new products.* Somewhat in line with the future-year planning theme is the task of reviewing new products as they come on the market. The logic for this is that very few business users consider an old IBM

PC/XT with DOS 2 to be a useful business computer. The pressures are even greater in the server and database world as users continue to increase their demands for processing and data access capabilities. The good news is that most vendors are willing to keep your mailbox full of literature and there are many third-party magazines that offer free subscriptions. Usually, it is not the availability of information that is lacking, it is the skill to effectively weed out the material that you need to read from the rest.

- ◆ *Keeping skills up to date.* With all the changes taking place in the products on the market and upgraded demands by users and developers, most DBAs feel the pressure of keeping their skills up to date. Most new products are fundamentally different from their predecessors (object-oriented and GUI development tools, for example). You remain of value to your organization by being able to provide the skills they need as they progress. It also helps your market value, which can always come in handy during your annual salary review or if you need to find other employment.

Just thinking about all of this probably makes you tired. Some of you may be worrying about what you are getting into. The only comfort is that there are thousands of others just like you who manage to get the job done. Remember the following key points to help you get through:

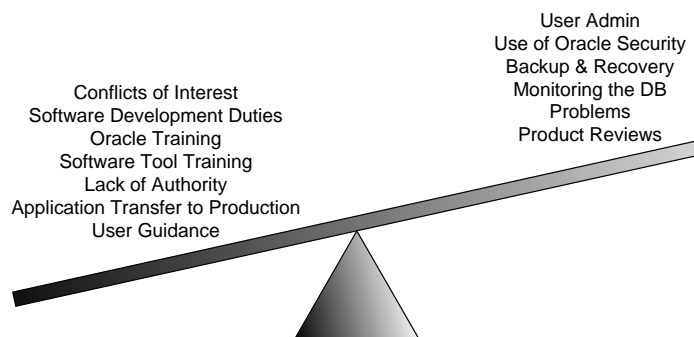
- ◆ *oad balancing.* Every request and ongoing support task needs to be classified as to when it really needs to be completed. Most requests come in with ASAP or “now” all over them, but I have found it worthwhile to ask the user the exact time it is needed. When you have this data and look ahead at upcoming routine tasks, you can begin the act of juggling them all to keep everything going. Although you may not keep everyone happy, you should try to do the tasks that will keep everything going.
- ◆ *Standards.* I love standards whenever they are possible. Such things as filenames, table names, use of special accounts to own database tables (more about this later) are all keys to making the job more efficient. These techniques are discussed later on in the book.
- ◆ *Watching for peak periods.* There are going to be workload spikes in even the best-managed shops. Such evolutions as installing a new release of the DBMS or upgrading an application can cause long hours of work that need to be confined to a relatively short period. The key is to clear all non-immediate tasks from the schedule during this period. I have found it helpful to publish via electronic mail (or even the old-fashioned paper kind) a note indicating that only emergency requests will be honored during these periods.

- ◆ *Being proactive.* I am often amazed how flexible many requests that say ASAP are when you actually explore them with the users. Another benefit of routinely asking about when the work is really needed is that people come to expect it and will usually work to give you some advanced notice. There are some situations wherein you are not given the luxury of controlling your workload. Perhaps you are in a shop that is afraid of outsourcing and you are under directions to jump when anyone says jump. However, if you can take a proactive stance, it usually helps you and your users get tasks done more smoothly.

DEVELOPERS ACTING AS DATABASE ADMINISTRATORS

Do not take this personally, but every time I see a developer acting as the DBA, I envision the robot on the TV series “Lost in Space” yelling “Danger, Danger.” It is not that developers are stupid—many developers are among the brightest people I have met. It is just that there is a fundamental conflict of interest in many developers who focus on getting that application—which they interpret to be the software that the users see—done on time. We computer types should be honest with ourselves that our on-time record in the past has not been stellar (for the industry as a whole, at least). When an industry has a bad reputation for being late, people tend to scrutinize your schedules closely. The developer acting as a database administrator has to perform the delicate balancing act shown in Figure 2.3.

Figure 2.3.
The developer/DBA's balancing act.



I have seen several examples of problems that have arisen with developers doing database shortcuts in development that made their systems a real pain in operations and that necessitated a large amount of rework. The intentions were good—get a system out on time that did what the users requested. The problems fall into three categories:

- ◆ Security is considered a burden during development and is planned to be implemented late in the development process after it is ensured that everything will work. This causes problems because security is implemented during a usually hectic period. Another problem is that sometimes the application design is incompatible with fundamental security concepts and this forces developers to revise their applications.
- ◆ The development database is small to save the cost on disks for the development computer and also reduce testing time. This results in a set of database queries that are correct in what they return; however, they do not return their data quickly when run against a production size database.
- ◆ The developers have no idea about the production environment on the target host computer. A production environment is a series of operating procedures and rules that are designed to ensure that the data center functions efficiently. You do not want to make operators remember a different set of procedures for each application. Production database administrators should be aware of these rules and ensure that they are followed. However, many developers are not exposed to this environment and may design a system that works correctly but does not fit into the production environment. Again, this results in headache and rework just prior to implementation.

Let me just relate one horror story that I observed and then I'll stop harping on the subject. This development was done by a nationally respected computer consulting firm. They were brought in by the business people to build a computer system to support a new business area into which the firm was moving. The computer support staff, including the database group, tried to provide rules for the developers, but they were told to lay off of them until things got closer to production. This was a turnkey project where the developers purchased the development and production computers, development software, and utilities.

When it came time for turnover, the database, system administration, and other support groups were allowed to review the application and were given the task of implementing it in the company's production computer environment. The database group found a system where nearly all users had DBA privileges (which allowed them to read or write to any table). The operating system shell scripts used to perform many functions that were performed using an account that had, in effect, root privileges (that is, system administrator on UNIX). When we tried to run the scripts using accounts that were permitted for users in productions, they did not work. Finally, after the security rework was completed, when these applications were moved to the standard production software directories, most of them did not work. As it turned out, the fully qualified filenames were hard-coded in the applications.

This was a rushed development effort and these people worked very hard to get applications ready that were needed to enable this company to start up the new business area. The developer who acted as database administrator, system administrator, and general software development guru worked many long hours on the project. Being late on the delivery of the product is not a good thing. However, there are times when certain shortcuts cause a lot of work later on in the project (when things are usually quite busy anyway). The software was implemented just slightly late; however, it was months before such basic DBA functions as security and backups were properly implemented. This came back to burn the project later on when data files (in the *production* system) were accidentally deleted by developers who were running with overprivileged accounts and the backups did not work. There were more than a few red-faced people with veins sticking out of their necks that day.

By the way, I have acted as both DBA and developer myself. Actually, I was in an engineering group doing some “real” engineering, acting as DBA and developing software. It is sometimes necessary for a developer to act as the DBA. Staffing budgets are not infinite and Oracle DBAs can be hard to find and are expensive (remember, we like the in-demand and high-paying part). This section merely points out what can go wrong. That way, when you work as developer and DBA, the software you develop will be on time, on budget, work perfectly, leap tall buildings, and so on.

To deal with a problem, it is necessary to understand the problem. Each situation may be slightly different, but the developer who is pinch-hitting as a DBA needs to confront the following problems:

- ◆ You have a conflict of interest. Attorneys and accountants are the ones who usually talk about this concept, but consider it in this case. You are a developer. Your compatriots are all developers. C is your native language, with English a distant second. Many years of management’s drilling into your head the importance of getting good software out the door on time and on budget are hard to shake. The DBA is traditionally an operational position concerned mostly with keeping a production system functioning smoothly. This leads to concerns about security, standard procedures, and other controls that promote reliability and operational efficiency, but make extra work for developers.
- ◆ You may still have a whole series of functional software modules assigned to you for development. These DBA functions are well and good, but where are you going to find the time to do them with all your other work?
- ◆ You have had a lot of training on C, C++, object-oriented, and event-driven programming techniques, but no training on being a DBA. Perhaps you were sent to the three-day introductory course on being a DBA, along with

the class on basic SQL, and now you are in charge of the database. This is a tough challenge, so let's hope they are paying you the big bucks to compensate for all your efforts.

- ◆ Finally, you may lack the authority that a full-time DBA may have. I have sat in meetings where the DBA group manager stood up and said that the application will not be accepted unless the development group does this, that, and the other thing. This manager reported directly to the director who also oversaw the development group—and everyone knew it. What do you do if you are a relatively junior programmer in a group of programmers, and no one wants to listen to your suggestions? To be honest, there is not have one answer that will apply in all circumstances. It is worth the effort to explain politely why it would be in the best interest of all to do what you are saying.

Let me challenge you now with another concept to consider as a developer/DBA. In the good old days, the software was everything. The DBMS is a collection of software that has been split out into an off-the-shelf package to save developers from having to write data transfer routines. However, starting with Oracle 7, you now have the option of storing part of your software in the database (stored procedures). In addition, the Oracle roles and enhanced system privileges associated with Oracle 7 make it easy for you to perform security functions in the database, rather than implementing security in every software module.

This leads to the concept of developing the whole system, which is a mixture of the database management system that you purchase, third-party development tools and applications, the software you develop, and the configuration in which you implement these products. The configuration is a combination of operating system and database parameters that you set to help provide service and control the application without having to write software. This is a new concept to many developers, but can be a real time-saver if implemented properly. The following are some examples of configuration parameters that can help build the application:

- ◆ Accessing grants to database tables and stored procedures in the database. This can save you from having to write a large amount of code to ensure that you have prevented unauthorized access to the database. In client-server environments, it may be your only way to provide security for the database tables when people access them using tools other than your software, such as Excel spreadsheets.
- ◆ Setting up Oracle accounts to use operating system authentication of the users. With this feature, you save having to maintain a set of Oracle passwords for the users. Instead, you indicate that if this user has made it onto the operating system, that is good enough for you. This is also impor-

tant in many environments wherein users refuse to log on to a computer more than once per day.

- ◆ Using standard Oracle utilities such as SQL*Loader, Import, and Export as opposed to writing custom routines to perform these functions.

To achieve this balance of the whole system requires a knowledge of what Oracle has to offer in the way of services. When reading the chapters later in this book, keep a special eye out for opportunities to save writing sections of code by using some built-in Oracle features. I hate to use a cliché, but this balance is working smarter, not harder.

The following is a list of things that developer/DBAs should consider for their systems. Because you have other assignments, you may not need to worry about the same things that the full-time DBA does. These functions should be considered for your Oracle instance:

- ◆ *Backup and recovery.* Imagine having to come into work and face your fellow developers after a disk crashed wiping out a lot of work in a test database. It would not be a pretty sight. Backup is just as important a consideration for development as it is for production. Try to convince your project team that they cannot afford to cut this corner to speed the development process.
- ◆ *User administration.* You just can't get around it. Someone has to generate new accounts. In the development environment, you have to balance holding other developers up by not granting them enough privileges against the risks of giving them too much power (everyone having DBA privileges, for example). Later chapters in this book that discuss user administration and privileges provide some guidance on this, but in the end, it is your judgment and experience that determines the privileges that you give out.
- ◆ *Monitoring the health of the database.* Space is usually the biggest concern in the development environment. You never have enough as developers create multiple versions of tables and try to get as much data in there for testing as possible. However, you should at least be roughly aware of the tuning status of your instance. You may save many hours of wondering why a particular application is not performing properly by ensuring that you have a properly tuned instance.
- ◆ *Dealing with problems.* Everyone hopes that all the problems with an application are found in the development environment. Sometimes the problems are in your applications. However, you may find that the Oracle software, the operating system, or other purchased products have bugs. You may have the pleasure of applying patches and working with vendor technical support to get the system stable so that your developers can do their work.

- ◆ *Providing guidance to users.* You may be regarded by your project team as the database guru because you have the title of DBA. If this is the case, try to obtain as much knowledge as possible about the DBMS and query optimization through training courses and books. It may mean more work for you, but it could save many hours of project team labor if you can quickly provide your compatriots with the answers that they need.
- ◆ *Reviewing new products.* Your primary job is to get a particular product out the door. Most development managers react negatively to the idea of changing the development environment in the middle of the project. However, you may reach a hard limit of one of the tools that you are using during the course of the project (for example, it can display only ten drop-down list boxes per screen). This is your opportunity to be a hero and suggest solutions. To provide these solutions, you need to keep an eye on the technologies that are out there. Perhaps it will be as simple as suggesting an upgrade to the Oracle 7.1 RDBMS product to take advantage of parallel query features when you are having query performance problems. In any event, the time spent obtaining this knowledge could come in handy on your next project.
- ◆ *Capability of transferring the application being developed into the production environment.* This was mentioned earlier, but it is important. If you, the DBA, do not think of at least matching the database configuration with that of the production instance, probably no one will.

In the right situations, a developer who understands the application design can set up the database to work well with the application. The points made in this section focus on ensuring that the database portion of the delivered system is production-ready with features such as security built into it. If the project DBA does not think of these things, they may not be included, because there is a very good chance that no one else on the project team will either.

SCIENTISTS, ENGINEERS, AND OTHER USERS AS DBAS

My first real experience as an Oracle DBA was using Oracle 6 on a VAX computer for a NASA project. I was in the engineering group responsible for tracking the engineering analyses and configurations for a very large space system. There were a large number of contractors located all around the world who had inputs into this system. I had just transferred out of the Information Resources Management group (yes, the functions were out-sourced to another contractor). I was the one who purchased the VAX and installed the Oracle database for another task, so I was the

logical choice to be the computer person in the engineering group. Of course, I had received absolutely no formal training on Oracle, databases, or any of the tools we were using. So I sat down with the documentation set and kept trying until I got it right.

This section addresses a broad range of people and systems, ranging from business people running a small store's inventory tracking system to a laboratory manager trying to track sample results. Although the applications vary widely in their function and needs, many of the DBA problems are similar. Specifically, I have found that these part-time DBAs share a list of common problems (I'm sorry, I mean opportunities to excel):

- ◆ Your primary area of training is not computers. Computers are merely a tool that help you get the job done. You may have had very little training on computers in general and there is not a large budget to send you to school on the intricacies of the Oracle RDBMS and your operating system. You have to pick it up on your own (which is why you purchased this book).
- ◆ You may not be allowed enough time to focus on the database. You have a lot of other responsibilities that relate to your chosen profession. Because computers and databases may be somewhat alien to you, it is tough to motivate yourself and find the time to sit down and become friends with your new responsibilities.
- ◆ Your needs may seem to be radically different from those of the traditional large corporate databases. Most of the literature is geared toward the multi-gigabyte systems with large staffs of people supporting them. You do not understand all this fuss about row-level security and two-phased commits of transactions.
- ◆ In many scientific and engineering Oracle instances, the data may be more of a loose collection of data as opposed to the focused order-entry systems common to large businesses. The design methodologies that try to link all the tables together do not seem to relate to your needs.
- ◆ You may stress different resources of the system than the common information systems. For example, you may perform a significant amount of floating-point computations as opposed to the large amount of input/output common to management information systems.
- ◆ You may not have a clear picture of what the application is going to look like when you start the project. Most businesses can define how they run their warehouses for people inventing an inventory processing system. In the case of many engineering and scientific projects, the results collected in the first round of experiments determine what you do in the next phase. Flexibility is very important in this environment.

- ◆ For small businesses and many smaller technical systems, you may be using commercial software that is from a small company with a small install base. You may find bugs that the few other users have not yet detected. These companies also usually do not have a large number of people to deal with problems that you find. You may have to be more patient and provide a little more support for yourself when you purchase these types of database applications.
- ◆ Finally, your view of the information may pose different problems than those faced by MIS applications. You may do a significant amount of processing on the data that you store, which makes it more difficult to validate the correctness of the raw data stored. You also may have to be concerned with the units for each of your data files.

Now you know the problems. What are some solutions? As is the case with the problems for the full-time DBA and the developer acting as a DBA, there is no one magic answer. However, there are a few things to consider that may help.

There are relatively low-cost alternatives to provide you with some training on tools and techniques. Books are one resource. There are interactive computer-based training courses that have the advantages of being able to quiz the student and enable the student to choose the topics being presented. There are also video courses on many computer subjects. Finally, even if management expects you to gain your computer knowledge on your own time and dollar, they may be receptive to support contracts with the vendors. When you buy contracts to provide service to keep the database going and receive updates of the product, you usually also get a number to call when you have questions. That way you can focus on understanding your applications and instance needs and have someone to call when a really nasty technical question or problem presents itself.

If you are pressed for time, see whether you can get the vendor or a third-party contractor to perform installations and upgrades for you. These can be time-consuming tasks, especially if you are not familiar with the product. The software installations also provide a number of alternatives for you to consider (size of various objects, tuning options, and so forth) The vendor of the product may have people who know the best alternatives off the top of their heads, which can save you a good bit of trial-and-error experimentation.

Although there may seem to be little in the way of literature discussing technical and engineering uses for computers and databases, it is out there. Many vendors have internal white papers that discuss tuning their version of UNIX for scientific applications or the settings for their Fortran compiler to match various analytical needs. Many of the vendors of laboratory software also have literature that explains how to implement their systems effectively. Ask vendors for such information (especially white papers, which usually contain the best really technical data and

also cost the vendors less to produce because they are not bound or sent to a publisher). It is always good to ask for other users of the product whose circumstances are similar to yours (for example, other firms using the system for finite element modeling). These people can be useful contacts when a problem comes up (they may not want to talk with you, but they might).

One of the greatest challenges will be ensuring that your instance is set up to run properly with your applications. The chapters later in this book address the basics of database tuning. When you receive a default Oracle instance, it is tuned to meet the typical environment of Oracle databases—a medium-sized business that focuses on transaction processing. Run the tuning checks described and see which parameters you have to modify to meet your needs.

When dealing with smaller companies, acknowledge up front that you may find more bugs and that it may take longer to get support. That is the price you pay when you need a very specialized application or you do not want to pay the price for the larger commercial applications. The keys to dealing with them is to allow more time in your project schedule to accommodate problems that you may encounter. Ensure that product support and references from other users are an important part of your vendor selection process.

Finally, with highly specialized applications that have a smaller installation base or perform a lot of calculations before displaying the results, it is especially important to validate the results before using new software modules. Remember the big fuss a couple of years back when preliminary data was presented on cold fusion and it turned out to be nothing at all? Try to construct a test data set that you can use to validate your processing software (for example, build a linear data set to ensure you get a linear graph, build a data simulator to calculate the expected response of a detector, and build a data set for the processing algorithm to process). Just do not be lulled into the notion that because the formula looks right and the computer is doing it that the answer will be correct (you may be the one to find a new bug with the Pentium chip).

Finally, it is time for a list of tasks that can be used by scientists, engineers, and other users who are acting as DBAs. Consider these when setting up your job description:

- ◆ *Don't forget backups.* Absent-minded scientists are allowed to forget to have lunch and dinner, but it could be disastrous if some chemicals in the air of your lab damage your hard disk and you have not backed up any data for two years. Don't laugh. I was on a major NASA project where people did a lot of modeling and analytical work on their PCs. Several of them had hard drive failures and the users responded with a great big "Huh?" when they were asked about backups.

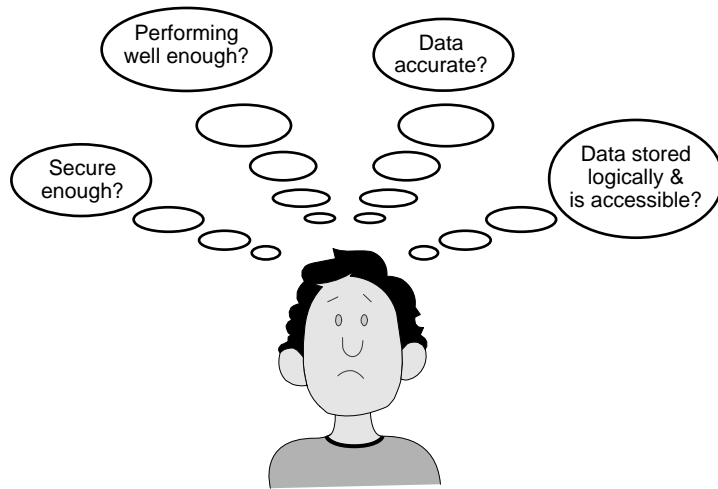
- ◆ *User administration.* You have to do it. Be careful of giving out DBA privileges. If all that your users need to do is create tables, give them resource creation privileges (see Chapter 11). The DBA has the power to delete anything within the database instance. A slight typo could wipe out a lot of data if the user has more power than experience.
- ◆ *Monitoring the health of the database.* Oracle is not typically tuned for scientific processing when it arrives. Your needs may be heavily weighted to simple input/output (as in the case of a lab sample result recording system), which will match the default settings well. You'll never know, though, until you run the monitoring scripts discussed in Chapter 30 (you do not have to type the scripts in—they are provided on the disk included with this book).
- ◆ *Dealing with problems.* This can be a problem in this situation. You may not have the budget to afford the expensive vendor support contracts. However, if you have an Oracle database as the heart of your lab or engineering system, you should really consider getting them. If all else fails, consider getting a local contractor who can provide support on an as-needed basis or contact other users who may have similar experience. Local user groups are a good place to start. If all else fails, you can post questions on several Internet newsgroups that relate either to Oracle, the operating system, or your technical field.
- ◆ *Control of data updates.* How many times do you collect a large run of data only to realize that you had one of the instruments set wrong? This can be a concern when you have a large database that contains data from many users and many processing runs. Specifically, when you realize the data is bad, how do you back out just the bad data of your database? Some solutions include previewing the data before it is inserted in the database (either on the instrument that is collecting the data or by storing it in a temporary table that contains data from a single run at a time). There are many possible solutions, and the one that you choose should fit within your unique environment. At least consider this function when you design your system.

DATABASE ADMINISTRATOR TASKS

The last three sections were not definitive job descriptions for each of the three types of DBA presented. That is just not possible because each DBA lives within a unique business environment and you have to adapt to your environment or face extinction (at least that is Darwin's theory). Think of those sections as food for thought.

The next sections present questions that are common to almost all database instances (see Figure 2.4). These threads are based strictly on business needs and should not be affected by whether you are a full-time DBA or not. Once again there is no one right answer that applies in every instance. When you ask the questions, you start on the path to finding the answers, so here goes.

Figure 2.4.
Common threads for DBAs to consider.



IS THE DATABASE SECURE ENOUGH?

I tend to look at safety and security in a very broad sense. Perhaps it can be described as the project manager's ability to sleep well at night, knowing that the data is safe and secure. Based on this concept, there are four areas that need to be considered for database security. The first is the traditional backup and recovery mechanisms that provide security against losing data due to media failures. The next area is security against unauthorized intrusion into the system and possible alteration of the data (hackers). The next area of concern is the accidental corruption of data by an authorized user of the system (the famous "oops" after hitting the return key). Finally, there is the consideration of the ability to keep the system running in the event of the destruction of the data center (or office in which your UNIX server sits).

The first of these areas is backup and recovery. I guess I like this topic so much because I always think back to the first time I did a major reconfiguration of an Oracle instance. I had very little experience with Oracle at the time, but needed to reconfigure the system to support growth. After work, my backup DBA and I sat down and gave it our best shot based on a plan that I had written up after reviewing the Oracle documentation. To summarize the experience, I fried the entire database

twice. You may ask how I could fry a database twice. The answer is simple, I followed the recommendations of so many books and made a complete backup of the database before I started monkeying with the system. I fried it twice, but was able to recover it each time thanks to my backup (the third time I got it right).

You may never have to do serious surgery on an instance until after you are a seasoned DBA. You may never make a mistake that causes you to lose data. Perhaps only kryptonite can harm you. However, you have to acknowledge the possibility that a disk drive will fail on you. Fortunately, Oracle has a strong system of backups built into it. There are two important points to make, though. First, Oracle gives you several backup options and you have to choose between them. Second, backup works only when you use it. Chapter 14 is devoted to this topic. For purposes of this chapter, remember that you need to develop a backup strategy that fits your needs (frequency of backup, type of backup, and so forth).

The next area of security to consider is protection against unauthorized access to your database. This is what most people think of when they think of computer security. There are entire books devoted to the stories of hackers who have penetrated the most secure computer systems in the world. Sometimes they use undocumented flaws in the host computer operating systems or utilities to penetrate systems that are well-protected by the local computer people. You have no practical defenses against people who know enough to exploit these arcane holes in your defenses. However, I would suggest that you consider the following easy-to-implement security points:

- ◆ Change the default Oracle passwords for your accounts. Everyone who reads the Oracle installation manual knows that system's password is `manager`. (I have been able to get into a large number of systems where the owners forgot the password by using these defaults.)
- ◆ Keep users' privileges to the minimum required for their jobs based on their roles. The idea of giving out full Oracle access to all tables and controlling what is done by an application is vulnerable to attackers, who guess easy passwords in a tool such as SQL*Plus and then use direct SQL access to review or destroy data.
- ◆ Provide especially good protection for DBA privileged accounts. Avoid the easy passwords. A DBA can obliterate the database and there is nothing to stop it.
- ◆ Because Oracle lacks a complex password protection scheme, use operating system passwords where possible and have the system administrator implement some of the more sophisticated operating system password protections (dictionary look up, password aging, password length protection). If this is not possible (in some client-server installations for example), consider assigning users some hard-to-guess passwords.

A third area of security to consider is protection against corruption of the data. I think of this as a person who has valid access to the database and makes an honest mistake. Unfortunately, much of the protection against these types of problems occurs in the design of the data tables and applications. Such features as confirmation screens to ensure that the user really wants to do something or logs showing data records deleted can assist in recovering accidental data deletes or overwrites. When there are situations where an operation is likely to wipe out a particular table, the DBA may consider routine exports that enable recovery of a single table that gets damaged.

The final security area to consider is the ability to continue operations in the event of severe damage to the primary host computer or the facility in which it is located. The common term for this service in the industry is disaster recovery. This typically applies to large businesses (who can afford this service) who cannot afford downtime. The earthquakes in San Francisco and flooding in Chicago provided many stories of businesses who had their data centers destroyed yet continued operations at backup computer sites. A small one-store operation has little reason to keep the computers up if the store is destroyed. It becomes a business decision as to whether it is worth the costs of having backup computer systems and routinely transferring data to these sites to ensure that they can come up and be ready to operate.

With all of this said, remember that this is taking a very broad view of security. Traditionally, security does not include backups or disaster recover (loss of the data center) concepts. The topic is broadened here somewhat to get you thinking about the broad scope of responsibilities that will be coming your way. Often, everything related to the database is considered to be the DBA's responsibility (especially when things go wrong).

DOES THE DATABASE PERFORM WELL ENOUGH?

The next question to task about your instance is whether it is performing well enough. If you polled users you would find a high percentage of them who yearn for better performance, better applications, and so on. However, we are not seeking perfection, we are seeking good enough. It is a business decision as to how much capacity and effort is justified when compared with the cost of upgrading it. I would argue that the performance of the system can be divided into three areas—response time, overall system capability, and processing accuracy.

The most popular complaint of users is that the “system” is not fast enough. In some cases they are right—a simple query takes too long due to the fact that the computer system is overloaded, the database is poorly tuned, or the query is poorly designed. In other cases, the answer is produced by having the database compute the average of a million rows of data scanned from a table containing ten million rows. In this case, the system may be doing an absolutely superb job with a very complicated query.

This brings up a very good question. What is fast enough? Computer people can argue this question all day long. Some spoiled techies say that the business could buy that new massively parallel processing machine for just under a million dollars to improve performance. Others gruffly recount how users had to wait until Mondays to get those answers before, because it took all weekend to compute the results. My favorite answer to the question is to ensure that you are doing the best you can with your current system and then turn it over to the business people. Once you have tuned your operating system and Oracle instance (see Chapter 32), you present it to business management that this is the best that can be done with the current equipment. Because you routinely review technology, you can tell them about new products that they can purchase to improve performance and what these products will cost. Then they have to decide whether it is worth it to the organization to speed up performance or tell people to work on something else while running this five-minute query. I have found that business people will argue all day with computer types (even when they understand nothing about query optimization or the proper setting for the redo log buffer cache), but when their vice president says that this is all that the business can afford so live with it, they accept it.

The next issue of performance to consider is what I call overall system capability. Perhaps your query response time is good, except at eight o'clock in the morning and one o'clock in the afternoon. When you look at your server's tuning and loading, however, you see no problems during this time period. When I have seen this situation, it turned out that the network was overloaded because people were checking and reading their electronic mail first thing in the morning and right after lunch. Although your Oracle instance and its server were functioning well, the whole system (which includes the local area network, wide area network, and perhaps other components) was overloaded. You may think that this is not your problem as a DBA. Unless you also have the duty of being the network administrator, I would agree with you. However, users see slow query responses at those periods during the day. It is not their job to understand network loading or the interactions of various computer systems, so they blame the database. It is your job to help track down the cause and prove that it is not an Oracle problem. Chapters 32 and 37 are designed to help you address this situation.

A final issue of concern to the DBA is processing accuracy. Perhaps in your organization this is completely the problem of the software development group. In rare circumstances you may run across bugs in Oracle or one of your purchased packages that cause incorrect results to be returned. It is not the DBA's job to fix the vendor source code that has the problem, but always try and keep informed of such situations and make sure your organization knows about any such problems. This is part of the overall job of monitoring the product and the industry.

IS THE DATA ACCURATE?

Many organizations have separate data administrators who ensure that the data itself is good. In some organizations, that actual business unit assigns a person to keep an eye on data quality. This combats the “garbage-in, garbage-out” problem that some computer systems have. If there is not a separate data administrator, you may be asked to ensure that the data is accurate in your system. The actual techniques used to ensure data quality may entail reviewing data feeds or ensuring that applications perform data entry checks.

IS THE DATA STORED IN A LOGICAL AND ACCESSIBLE MANNER?

You've got good data. You have a magnificently tuned Oracle instance. Can everyone find what they need? Just something to think about when you are setting up your system. If you are running a purchased application, this is probably not a concern. However, if you are trying to set up a decision support system where business analysts will be designing their own queries, you may not want to store data in a fully normalized format. One technique that I have seen in these circumstances is duplicating data in multiple tables to prevent them from having to construct complex joins. You also can construct database views that contain complex join logic within a logical structure that can be queried as if it were a simple table. Another solution is building summary tables that prevent them from having to build complex code to get commonly used numbers. Finally, always try to use description names for tables and columns. This is not the mainframe or DOS world where you are limited to eight characters.

INTERFACING WITH SYSTEM ADMINISTRATORS AND OTHER SUPPORT STAFF

One of the most difficult tasks when traveling from organization to organization is establishing interfaces with other computer support staff. Perhaps it is a function of industry trends toward out-sourcing and down-sizing, but many people are really sensitive about what they perceive to be their job description. This often happens in shops that are installing UNIX computers running Oracle after having had mainframe computers for some time. The UNIX world tends to enable support people such as DBAs to do a number of things, such as software installation, backup, and automatic job scheduling that were relegated to the tech support and operations groups on the mainframe. Many of these mainframe folks honestly believe that even

the thought of anyone else doing this stuff is a threat to system integrity. Your challenge is to understand enough of what Oracle does and explain what needs to be done well enough that a reasonable compromise can be found between the two worlds. This is not an easy task. Many of the managers of these other support groups are used to being gods who pronounce judgments that cannot be violated.

What are the areas that typically need to be worked out between support groups? The first is usually the one that installs the Oracle software. Oracle's installation procedure is based on having the system administrator load the software from tape and then run a script that gives the oracle account the necessary permissions to do the remaining work. The Oracle DBA configures the Oracle software and builds the databases. The problem can come up when you have system administrators who are not familiar with Oracle and therefore have difficulty choosing between the various options that Oracle presents on installation. I prefer to either install it all myself or sit down with system administrators before beginning the installation and go through the book, marking up which options to choose, where to put the software, and so forth. Often I type up a procedure extracted from these manuals showing the options to select that. Some of these people live for written procedures, so this makes them very happy. I almost always prefer to create the instances myself, but if you have to, you can write up a procedure so that the system administrators can do it.

The next common area of negotiation is backups of the Oracle database and software. Backing up the Oracle application software should not be a great concern as long as someone is doing it. The real concern comes in the backup of the database data and log files. Most operating systems enable the system administrator to grab any file on the system and copy it to tape. However, if you have not issued the proper commands to the Oracle database, the data file that is copied to tape may be inaccurate. In the end, it does not matter who issues the actual commands to perform the backups (a job scheduling utility such as `cron`, an operator, the system administrator, or the DBA). What *does* matter is that the commands are correct from a database point of view. Chapter 14 discusses backups in more detail so that you know the correct commands to back up your database.

A third area of concern is the monitoring of the Oracle database and operating system. Many of the database monitoring commands require a fair amount of Oracle knowledge and DBA access. These are almost always used by the DBA to monitor the internal workings of Oracle. Sometimes there is an issue as to the ability of the Oracle DBA to access operating system monitoring commands (such as `sar` under UNIX or `monitor` under VMS) to monitor operating system performance. Part of the tuning process discussed in Chapters 30 and 32 requires you to determine whether the problem is with Oracle or its host computer. I always prefer to have access to this information because I do not have time to run around and find a system administrator when a problem occurs. However, you may have to in some organizations where they do not give out access to these utilities.

Finally, you need to come to an agreement regarding the processing of trouble calls. In organizations where paranoia runs rampant, the first thing done when a problem occurs is finger-pointing at every other group in the data center except your own. Because Oracle is often the new guy in town, it is easiest to say that it is that new-fangled, non-MVS software package that is causing all of the problems. Perhaps the problem is with the Oracle instance. However, you'll never know unless you collect some data and determine the true cause. Chapter 34 discusses how to troubleshoot various types of problems that may come up with an Oracle database. You need to consider getting an agreement with the system administrators and network administrators as to how you will work as a team to solve a problem with the overall system. Perhaps the burden of proof rests on your shoulders, but you need to define the points at which you get the other support staff involved and what evidence you need to present to them to prove it is a problem with their part of the system.

This section could not possibly cover all of the issues that may arise when you figure out which tasks belong to the DBA and which belong to other support staffers. It's merely a section to stimulate your creative juices and thought processes. If you come to these agreements beforehand, you can save an enormous amount of time arguing over who is supported to do what when it comes time to do the work. All DBAs, especially those who are administering the first Oracle instance on a UNIX platform in a legacy computer shop should stay calm and do their homework. When you need to suggest changes to the way things "have always been done," you should be ready to pull out chapter and verse from the Oracle manuals that show why backups have to be done differently than they are done on IMS. As with so many other tasks that involve people, just try and do the best you can to get a working arrangement that enables you to do your job.

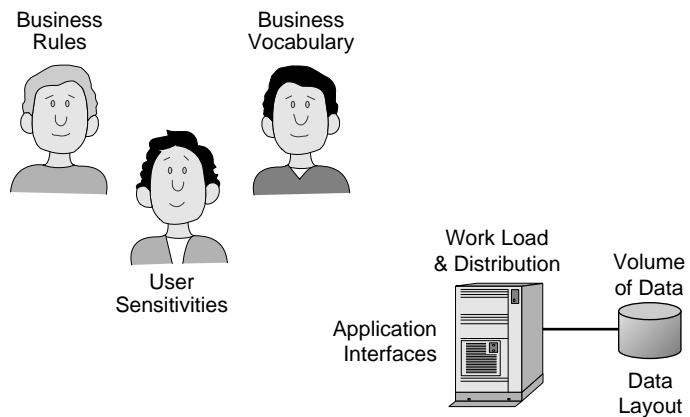
UNDERSTANDING THE APPLICATION NEEDS

Many of the computer support staff that I run across can tell you more about an overall business process than the workers up in the business unit. Many business groups are composed of a number of specialists who focus on one particular task. The computer systems are usually designed to support a number of tasks in the business process and therefore have to be concerned with how the little tasks integrate to form the overall business process. This section is designed to present a quick overview of what the DBA should be thinking about to understand the business process well enough to properly support it (see Figure 2.5).

The first challenge faced by most computer staff members supporting an application is learning the vocabulary of the business people. The vocabulary of computer types is perfectly understandable (RAM, ROM, DRAM, instances, and SGA), of course, but some of the business words are a challenge (gross accounts receivable and net rate of return). I have sat in a number of meetings between users and computer types

where they keep talking at one another but there is no communication because they do not understand each other's language. I remember one project where I had a very bright young programmer for whom I had to translate user's requests into field names and programming loops to enable him to go do his work. This is especially true when you are dealing with business people who are not comfortable with computers. If you do not speak in the terms that they are familiar with, they get very nervous, so try to learn some of their terms.

Figure 2.5.
Understanding the application needs.



DBAs should also understand the sensitivities of the user community that they are supporting. Having traveled to work with a number of different types of businesses, I have found that each group has a certain set of sensitivities. Perhaps the users had a lot of data lost on the old computer system, so they focus heavily on backup capabilities. Bankers are especially sensitive about having audit trails to track the movements of the money in addition to the total amount in a given account. You need to understand what these hot buttons are for your users so that you can build solutions into your plans for them. The users are going to ask for them anyway, so why not anticipate them to avoid rework later on to your plans.

There is another challenge closely related to these first two areas. The DBA needs to understand the business rules associated with a system. Perhaps you do not need the detailed understanding of the system analyst who is building the applications. However, many of the tasks that you will be asked to perform, and how the various applications interface with each other and the database, will be determined by business rules. A fair understanding of the high-level business rules helps you understand what the users and developers mean. It also helps you piece things together when determining the loading on your system. Your job is to fuse these business needs with sound database design techniques to form a system that meets their needs.

Now, for you pure techies, on to the technical challenges. One of the obvious ones is to understand the volume of data that you need to manage. The simplest approach is to take the number of bytes of data added to and subtracted from the system each month to calculate the amount of space you will need in the future. This may work on some systems, but not many of the ones with which I have worked. The real problems with space come in when developers are planning to field a new application in three weeks that needs an additional 10G of storage space or the business people have just acquired a small company that is going to double the number of orders you are processing per week. You have to ensure that everyone in the organization is keeping you informed of these types of changes. Hopefully they will tell you far enough in advance of these changes so that you can acquire additional disk space and reconfigure the instance to support the new volume.

The next concern is the amount of work that will be performed and how it will be distributed throughout the day. Your goal would be to try to move jobs that are flexible into periods of low load (that is, run your backups and auditing reports at night). You should also be sensitive to the fact that it may not be the number of jobs running that kills performance, but the type of jobs. If you have two jobs that really hit a particular tablespace, try splitting them at separate times. Much of this you will pick up by experience (looking to see what is running when things are slow). For now, just remember it is something to keep in the back of your mind when developers are turning new applications over to production.

The DBA also needs to understand what is in the various data tables and how they are laid out in the system. This may seem obvious to many DBAs, but for those who acquire packaged applications such as Oracle Financials, extra effort is required to figure out what is stored where. The first thing to check is whether the tables and indexes are located on separate disks for those systems where input/output performance is critical. You can print out a list of the tables versus the tablespaces and keep that around to help when you experience performance or space problems.

Finally, the interface to the various applications should be well understood. For example, you may have a database instance that is up and working perfectly but users cannot access it. If you did not understand that these users access the system using client-server applications, you would not think to check whether SQL*Net is working correctly. Some client-server installations can get even more complex when users access an application server computer that accesses the database server computer. The point is clear, the developers or vendors need to present you with an accurate picture of the architecture before they expect you to maintain it.

Think of the topics in this section as a list of knowledge to gather over time. This is a lot of information to keep straight. For those topics that you do not know off the top of your head, at least figure out where you can get that information (database

queries, printouts, and so forth). You do not need this data on day one to get your instance working. You can gather it as time permits. It is worth gathering because having it available will save you time in the long run—both in dealing with problems as they come up and in preventing some crises from occurring.

SUMMARY

This is a challenging chapter. Some of the topics may seem alien to pure techies who do not like to think about politics and management issues. However, pure techies tend to be the ones who experience the most problems with unrealistic job descriptions formed by others who do not understand what is involved with each of the tasks.

Remember, there is no one, true picture of an Oracle DBA's job. You need to evaluate your individual situation and select those items from the lists presented that make the most sense for you. Finally, do not expect to get everything in the job description correct the first time. Give it your best shot and expect that your job will evolve over time as changes to your organization and database products evolve that require different forms of support.

- 
- Computerized Data Storage
 - The First Databases
 - Relational Databases
 - Oracle's History
 - The Current Database Market
 - What Next?

CHAPTER 3

History and Development of Databases and Oracle

OK, this is not a history book. However, a brief chapter on the history of databases provides perspective. Sometimes you wonder why something has been done a certain way. The way things were done in systems that preceded Oracle is often the reason behind the way Oracle does things. For seasoned computer veterans, this chapter may seem irrelevant. However, some of the new DBAs out there have never known life without a Windows-based PC in their house and think punchcards are something from the Middle Ages.

COMPUTERIZED DATA STORAGE

In the beginning, all information was entered into computers manually by such devices as switch registers (or even rewiring). The first computer that I worked on in high school had disk drives, but you had to toggle in a boot sequence that would read paper tapes that started the operating system from the disk drives. The disks contained only operating system data and were extremely small. I can't remember the exact size, but they were in the kilobyte range and cost a fortune by today's standards. Before any younger readers make any remarks, that was only the 1975–1977 time frame.

All user programs and data were stored on card decks. The programs and data were placed in different sections of the card deck. Heaven help you if you dropped a large card deck on the floor. (I've seen it happen, it is not a pretty sight.) As you can imagine, data stored in this format was extremely limited by today's standards. Even cabinets full of large boxes of data cards would fit on most PC hard disks today. The very high charges for computer time placed strong limitations on the complexity of the applications. I remember spending thousands of dollars developing and running software that would fit on the smallest of PCs today.

However, those of us who earn our living in the computer industry can thank those early systems for showing what was possible with computers. Businesses, government, and universities became hooked on computers and wanted to see what more they could do with them. The high cost of computers gave the vendors at the time money to spend on research and development. Significant improvements were made in the storage capacity and costs of key hardware components, such as central processors, disk drives, and terminals. This led to a change in operating systems. They started to support storage of software and data on disk drives, including the operating system.

From a data processing point of view, this enabled application developers to start storing data online in files. As disk drive costs fell, it became easier to justify purchasing additional disk space to store more data. This enabled companies to perform more analytical work comparing this month to last month or looking up information on an order that was shipped last month. The addiction of organizations

to computers and computerized data storage continued to grow. This increase in technology and reduction in costs led to a new way of thinking in the computer industry.

When computers were first introduced, the hardware costs were enormous. The cost to service and program these computers was relatively small. People were forced to write programs and perform services that made the computer's life easier. I remember one of my former bosses talking about what they had to go through to modify the assembler language software for the onboard computers that were used in the Apollo program when the Apollo 13 accident occurred. Of course, as with all parts of large programs at the time, they had a vast sea of programmers and engineers to call upon if needed.

When the price of computer hardware started to drop, the equation changed. With disk drives being relatively cheap and computer processors becoming more capable, the cost of human beings became significant. In many shops, the people costs started to exceed the computer hardware costs by a wide margin. This change had many effects. First, more sophisticated programming languages such as COBOL and FORTRAN began to replace primitive assembler-like languages. This increased programmer productivity. As computer hardware costs continued to fall, companies began to search for other ways to lower the software development and people costs for applications.

One of the techniques that they developed was the database management system. When disk data storage first became available, the files had only primitive means of separating different data components. Most file systems provided a marker that showed the end of a line of data and another marker that showed the end of the data file itself. Everything else was left to the discretion of the programmer. Most applications divided lines of data into fields of data by specifying which columns corresponded to the fields. All knowledge about what was in the fields and how the fields related to one another was locked inside of the applications.

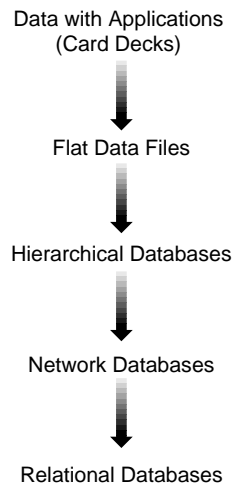
This made sharing of data files between applications difficult. Both applications had to understand the scheme and any changes to the data scheme required changes to all applications that accessed this data. Each application also had to include algorithms for searching the files for the line of data that you wanted, algorithms to insert a new line of data at the end of the file, and so forth. This was wasteful because you had to write the same software over and over again.

This is where the first database management systems came in. They were not very fancy; however, they provided an interface to a common set of algorithms to perform the most routine data storage tasks. Some would debate whether these systems could be classified as true DBMSs or whether they were merely subroutine libraries. However, they soon evolved and became systems that not only provided common access routines. They soon started to improve the way data was stored.

Consider the flat files or punch cards originally used to store data. They typically had to be read sequentially. Combined with limited computer memories at the time, the current row was almost all the data you had available to you at the time you performed a calculation. Therefore, you had to make sure each row contained all of the data that you would need, even if that meant repeating the same value in hundreds of rows of data. In most applications, this turned out to be repeating many fields in all of the rows of the file. Many of the definitions of what a “true” database is require that the data be stored without redundancy. Many of the early database management systems included this feature of enabling you to store data in many different parts and then linking them together with the database.

There were a number of steps in the evolution of data storage. Many of these steps were experimental and never caught on in the computer industry. For purposes of our discussion, let’s break down the evolution of commercial data storage to date into five eras, as shown in Figure 3.1. This discussion discusses only commercial data storage systems, not those experimented with in the laboratory or those with highly specialized niches. The other key words are “to date.” There is strong movement in the database industry to expand beyond the relational model and move into the capability of storing general forms of data, such as images and video clips.

Figure 3.1.
Evolution of data
storage to date.



THE FIRST DATABASES

This section presents a brief introduction to the early database systems. Some programmers simulated the functions of a database management system within the software that they designed to work with flat files. The key to a database management system was that the programmers could concentrate on programming the business functions while leaving much of the data management issues to the DBMS.

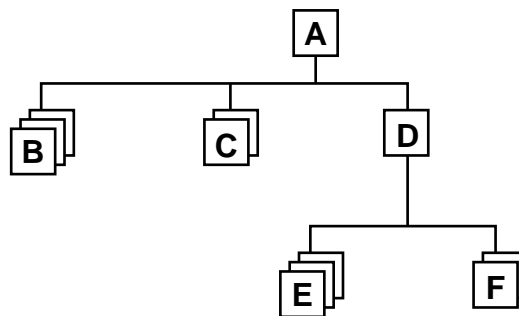
There were also some other features that most flat-file systems had that should be considered:

- ◆ Data definitions were in the applications software making changes difficult in large application systems.
- ◆ There was no control mechanism if two users tried to update a single record at the same time.
- ◆ Data was redundant.
- ◆ All access to the database needed a specific computer program to be written. This program needed to be customized to handle the way the data was physically stored in the flat files.

The first real commercial database management systems were based on the hierarchical data model (see Figure 3.2). You can think of this system as storing data in a tree. The parent record has one or more child records. For example, a person record may be considered the parent record with name and social security number. There may be multiple child records for such things as addresses (the person may have moved several times), previous jobs held, and so forth. A big commercial implementation of this strategy was the IMS product. There were some limitations with hierarchical storage systems that spurred the development of the later data models:

- ◆ A child could relate to only one parent record. This promoted some data redundancy.
- ◆ To find a child record, you had to navigate through the parent records.

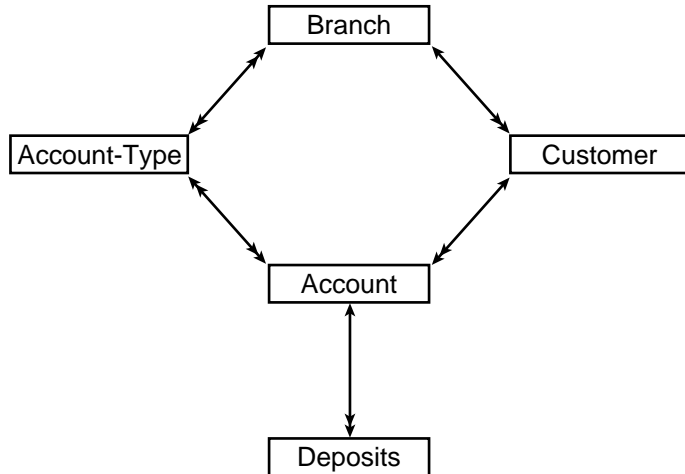
Figure 3.2.
The hierarchical data model.



The next major data model developed was the network (see Figure 3.3). This model was developed in the 1960s by a series of conferences on data language standards. Another reference to this standard would be CODASYL. It was designed to eliminate the need to store redundant data. A given record can have multiple children and multiple parents. Although this network provides flexibility with

parent-child relationships, the networks for larger applications became very difficult to navigate through (for both programmers and the DBMS itself).

Figure 3.3.
The network data model.



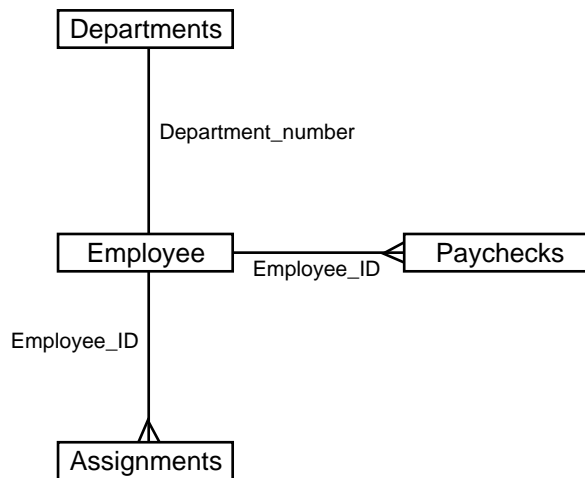
RELATIONAL DATABASES

As online data storage demands continued to increase in the late 1960s, research projects were conducted to find better ways to store data for business use. There were many models produced, but the one that seemed to stick in the minds of the user community was the relational database model. This model was first documented in a paper from an IBM research project in 1970 by E.F. Codd. It must have been difficult for IBM management that their research project produced the relational data model, but Oracle had a commercial system on the market several years before IBM released its DB2 product.

The relational data model is an extension of some of the concepts of previous data models. It is designed to enable flexible access to the rows of data that you want to access without forcing you to go through a large number of nodes in a network. Actually, much of the initial theory was based on mathematical set theory. It is always nice to see university theory being applied to a useful purpose. The basics are that data is stored in a series of tables. Each table is designed to store data about one type of thing (also referred to as an entity). The columns of the table represent properties (or attributes) of the entity. For example, consider a table of people working for a company. The entities are people. You cannot store part numbers or garden plant varieties in this table, only people. The columns of data are attributes of the people, such as birthdate, social security number, and so forth. You cannot have columns that store the gross profits of your company or the number of fish caught in a particular stream.

The relational concept comes in when you try to link these tables together. You can designate that certain columns of data will have the same meaning in multiple tables (see Figure 3.4). For example, the social security number in a table listing the people working for a company will have the same meaning as the social security number in the table storing all paychecks issued. This is a one-to-many relationship. The company will issue paychecks every week or two to employees as long as they remain on the payroll. There may be a few employees who quit after one week, but that is not a problem with this model. You can also have one-to-one and many-to-many relationships in the relational model.

Figure 3.4.
The relational data
model.



Oracle and DB2 are not the only commercial relational database management systems. In fact, most of the commercial database management systems that you will come across in business today are based on the relational model. The following are some examples of popular relational database management systems:

- ◆ dBASE
- ◆ Foxbase
- ◆ Microsoft Access
- ◆ Informix
- ◆ Sybase SQL Server and Microsoft SQL Server

dBASE made the relational model popular. Although it is easy today to criticize its lack of development efforts after versions 3 and 4 of their product and the amount of time they took to bring out a Microsoft Windows product, I would give them credit for training a large number of programmers on relational database theory. It was so easy to get a copy of this product and produce complete applications on a PC.

Because developers did not have a database administrator, as in larger systems such as Oracle, they had to think about optimizing queries, indexes, and backup/recovery functions—subjects that many developers could ignore.

ORACLE'S HISTORY

Oracle was founded in 1977 by Larry Ellison, Ed Oates, and Robert Miner. The original name for the company was System Development Laboratories. It was a consulting firm (nice to know consultants at least have a chance at a bright future). In November 1976, the *IBM Journal of Research and Development* published an article on relational database management systems. The owners changed their company name to Relational Software Incorporated. Their goal was to produce a database management system that implemented the published theories on relational database management. Oracle produced its first RDBMS in 1980 (IBM released DB2 in 1984). A key part of its philosophy was to market the product on a wide variety of computers, including mainframes, DEC VAX computers, and many of the smaller minicomputers that existed at the time. This clearly differentiated it from IBM and DB2 (DB2 ran only on IBM mainframes until just last year). Perhaps part of Oracle's success could be attributed to the success of minicomputers in the early 1980s and UNIX servers in the late 1980s.

Early versions of Oracle ran on Digital Equipment Corporation PDP-11 computers. Oracle decided early to write its software in the C programming language (back when C was a new toy on the market). This allowed it to port its system to DEC's VAX computers (including the UNIX operating system). Much of Oracle's early business came from the popularity of the VAX computers. The company name was changed to Oracle to make it easy for customers to associate the company with the product.

By version 4 of the Oracle RDBMS, the company had ported the operating system to run on IBM mainframes and personal computers. Version 5 supported a new concept known as client-server computing. Version 6 started the migration of the product line to support large, business-grade databases. This trend continues with version 7, which provides some administrative enhancements as well as a number of functional improvements for computers with multiple CPUs.

The following are some of the other developments in Oracle's history:

- ◆ 1984—First PC Oracle database
- ◆ 1987—Oracle Financial Applications release
- ◆ 1988—Computer aided software engineering (CASE) tool
- ◆ 1988—System integration subsidiary
- ◆ 1988—Support for multiple processors (version 6)

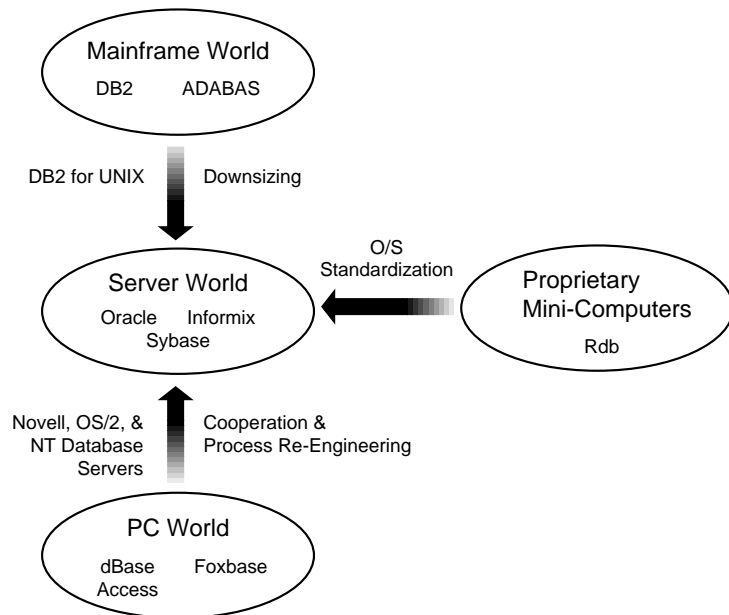
- ◆ 1990—Object data types included
- ◆ 1992—Support for distributed databases (version 7)

THE CURRENT DATABASE MARKET

Several years ago, almost every database vendor with the exception of Oracle was happy to make money in its own world. PC database makers such as Borland (with dBASE) stuck with single-user PC databases. Mainframe vendors such as IBM stuck to the mainframe. Most of the UNIX server vendors, with the exception of Oracle, stuck to UNIX server versions of their products.

Perhaps it was Oracle's success in offering its products across a number of platforms. Perhaps vendors recognized that industry trends in down-sizing made the UNIX server market more attractive. Perhaps it was some of the push by Microsoft and Novell, combined with the rapid increases in power of PC servers that made these platforms more respectable as database servers. Whatever the reason, there have been database vendors who have pushed to offer their products across a wider range of product lines (see Figure 3.5). Even IBM now has UNIX- and LAN-based versions of the DB2 product that remained only on the mainframe for a decade.

Figure 3.5.
Competitive trends in
the database industry.



Oracle continues to lead the relational database market, at least for now. You will see a number of figures published. Some talk about number of copies sold (which favors the cheaper PC database). Others talk about dollar value sold (which tends

to favor the larger systems). Still others quote specific figures, such as the number of copies sold on the Data General Aviion platform to universities. Most of the figures indicate Oracle is the overall largest RDBMS vendor with roughly one-third of the market. IBM has about one-quarter of the overall market. The other vendors split the rest and there are a large number of them out there.

WHAT NEXT?

For those of you who are planning on making the care and feeding of databases a career, it is important to consider where things are going in the industry. Then, you will be there when the industry gets there and be positioned for employment. There are two trends in the database industry that you should at least keep an eye on for the future. The first is the tendency toward object-oriented databases. The second is the multimedia database.

Most of the “hot” development tools in the industry talk about how they are object-oriented. Most of them have at least some component of object-oriented technology in them, but most are not fully object-oriented. Without getting into a discussion about the fine points of object orientation, consider it to be the merger of data and software (that is, processes) together. You can select data and have a list of things that you can do with the data (methods) pop up for selection. There are several interesting object-oriented databases on the market today. They suffer, as most small products do, with having to interface to a wide range of systems and overcome marketing hurdles. A more serious problem is the fact that most major relational database vendors are migrating their products to include object-oriented concepts. Oracle’s capability of storing software within the database (stored procedures and triggers) is a first step in this direction.

The other trend seems to be hotter at the current moment. Almost everyone in the computer industry seems to be caught up in the multimedia craze. Part of this may be fear that they need to get to market early before the market is completely fixed. The current competition between cable television and telephone companies in the United States to provide the next generation of television service represents an enormous market. One of the key components of this market is on-demand TV, which requires an effective means of storing and transmitting video signals in an automated fashion. Oracle has been pursuing this market for some time, betting heavily on massively parallel computers to provide the computer power needed for this application.

SUMMARY

This book is devoted to being a practical guide for DBAs, and the background on the database world may be useful to some who did not grow up in this environment. With this general industry background, you can move on to the Oracle environment itself in the next chapter.

This chapter is going to be a bit of a challenge. Some organizations use the Oracle database and Oracle tools to develop local applications. Others buy third-party complete application packages that are based on the Oracle database. Still others implement client-server applications using the Oracle database and Powerbuilder on PCs. This chapter cannot present all the possible configurations. Instead, it offers some of the common types of environments that I have run across that you can use for a reference when forming your environment.

A definition of environment is in order at this point. A computer system is composed of a series of hardware and software components working together to service user needs. Some of the hardware and software may be dedicated to making the network communications functions possible. Other components may provide tape storage services. In this book, the term environment means the sum of all of the components in the system that surround the Oracle database and how they are connected.

In this chapter you will explore the different types of environment and the components commonly found with the Oracle database. A brief discussion of integrating third-party (made by a company other than Oracle) products into the environment is also presented. In the last of this section, I will discuss some of my experiences trying to get all of the pieces to work together. Remember, you cannot be a top-ranked DBA if someone purchased a version of Oracle for the wrong operating system.

WHAT IS ORACLE?

Perhaps the most fundamental answer to this question is that Oracle is a corporation headquartered in California that makes a series of computer software products. As discussed earlier, there are a number of products, ranging from the flagship product (the Oracle relational database management system) to complete, industry-specific applications such as Oracle Financials. Go back and take a look at Figure 1.1 in Chapter 1. It is a pictorial representation of the many ways that different people look at Oracle.

Many DBAs see Oracle as a relational database management system with a bunch of other stuff scattered around it. These other tools, such as the Oracle Forms development tools, are just ways of getting data to and from the database. Many developers, on the other hand, see Forms or Powerbuilder as the focus of their lives. A DBA, especially one working in more challenging environments such as client-server, should avoid taking a narrow view of the Oracle system. In most environments, it is important for the DBA to understand all the pieces that are involved in transferring information from the database to the end users. This is because many system and network administrators tend to take a specialist attitude. They do not understand the workings of databases or even each others' responsibilities. Perhaps

they are supporting a large number of applications in addition to the database and do not have the time to learn about each application. Far too often I have received that trouble call that the “database is down” only to find out through troubleshooting that it was a network or operating system problem.

Therefore, I would argue that the Oracle world, as seen by the DBA, should include all hardware and software products needed to connect users with your database. You do not have to know the intimate details of how the silicon chips on the communications cards are produced. Just a knowledge of what the components are and what they do goes a long way toward helping you. The troubleshooting chapters later on take this system knowledge, combined with the results of a few monitoring utilities that are commonly available, to help track down where in the overall system the problem lies. You will probably not learn this all over night, but rather pick up bits and pieces as time goes on. For example, I take every opportunity at a design review meeting to have the specialists draw out their components on a sheet of paper. Then I place that paper in my files for when the problems start.

ALTERNATIVE ARCHITECTURES

There are a number of very thick books devoted to computer architectures. They discuss the various network topologies, CPU chip types, and many other fascinating topics (if you are into that stuff). This section distills this material to what the DBA needs to understand and mixes in the Oracle specifics that are not discussed in these general books. Part of the difficulty in this topic is that there are so many ways to classify the architecture. Each type of analysis is valid and important to different people in the computer industry. This section presents system architectures by the following three criteria, which seem most applicable for Oracle DBAs:

- ◆ Computer and communication system architecture
- ◆ Usage of the database (a logical architecture)
- ◆ Processing architecture of the host computer running the Oracle RDBMS software

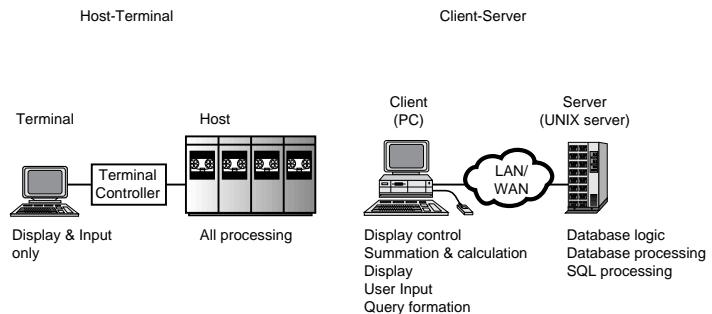
There is a major impediment to communications among computer professionals—excessive vendor loyalty. Loyalty is usually perceived to be a noble trait in individuals. I have never worked in sales and have also never worked for a computer vendor (consulting is as close as I have come to this). I have no great vendor loyalty other than the one that helps me get my job done. I’ve even installed and worked with Informix and SQL*Server databases.

It is difficult to deal with the almost violent reactions that some people have when someone presents an architecture or products that do not match those of your vendor. Many people see it as admitting a mistake to consider HP or IBM UNIX boxes when

they recommended purchasing that mainframe five years ago. Remember, recommendations are based on the products and information that were available at the time. If a vendor comes out with radical improvements to a product line or changes architecture, consider using it in your next system. On the other side of the coin, there are techies who want to jump to every new product that catches their fancy. This often results in an architecture that is almost impossible to maintain. Your challenge is to try to forecast which company has the best chance of meeting your needs, both now and in the future. You also have to be willing to look at your own decisions with a critical eye. Try to keep an open mind when this book discusses architectures that with which you are not familiar. With the way corporations are going these days, you may get laid off only to get a much better job working in one of these new architectures. It is all just a bunch of ones and zeros.

Now back to the subject at hand. Computer architectures can be classified according to the computers and communications systems used. This is one of the hottest areas of debate and change in the computer industry today. This section doesn't focus on individual vendors, but, instead, refers to the size, type of operating system, and network access technique. Figure 4.1 shows the old-style host-terminal and newer client-server architectures.

Figure 4.1.
Computer and communication system architectures.



Let's start with what more people are probably familiar with, the host-terminal architecture. In this architecture, the thinking is done on a relatively large central computer. This computer runs the user applications and the Oracle database, and provides all other computational and communications services. The terminal has two simple functions in life. It displays the characters that the host computer tells it to display and transmits the keystrokes entered by the user to the host for processing. Some terminals have a small amount of intelligence to transmit an entire page of information at a single time and other such features. However, for our discussions, the host-terminal architecture puts all the brains in the central computer.

Oracle originally designed the RDBMS and most of its development tools to function in this environment. You build forms and reports that run on the host computer while accessing through host communications facilities to the Oracle database. The terminal displays were primitive by today's standards and you had to remember a large number of function keys to make your applications work. Oracle has enhanced its host terminal products to provide a somewhat better interface. This architecture does have the advantage of being cheaper per desktop, although the applications may not be able to provide the same level of functionality.

In contrast to the host-terminal architecture is the client-server world. Almost every computer salesman out there is selling something that they would classify as client-server. It is a hot topic in all the magazines and a lot of people want it. The basic concept is simple enough. You distribute some of the thinking work from the centralized computer (known here as the server) to the desktop computer (usually a PC) that is referred to as the client. The difficulty comes in when you listen to vendors discuss how much of the processing should be distributed to the client. This ranges from products that transfer some of the logic as to how to display the output data to others that leave only the core database functions on the server. One key point to remember is that client-server almost always presumes some form of computer networks (not just terminal servers). The network is an extremely critical part of this architecture that is often overlooked when designing these systems.

The second way that you can cut the system architecture is usage of the database. This is a logical view of its function and processing as opposed to the more physical layout of terminals, PCs, and host computers in the last discussion. This architecture tends to be more closely tied to individual applications and is a starting point for the overall system design. It tends to be closely tied to the structure of tables and the database resource demands of the applications. For purposes of this discussion, let's divide the many possible uses of the database into three categories:

- ◆ Operational or transaction processing systems
- ◆ Data warehouse systems
- ◆ General-purpose Oracle systems

The first usage class is probably the most common use of databases in industry today—the operational or transaction processing system. It is typically categorized, in relational databases, with highly normalized (many tables, each containing a specific type of information) and efficient tables. These tables are tied together using key fields (fields that have the same value in multiple tables that can be used to show a relationship between the rows of data). The focus of the system is efficient entry and retrieval of a relatively large number of updates and insertions. A retailer order entry system is a classic transaction processing system.

Some of the key points to consider about operational systems within Oracle are reliability, recovery, and transaction resources. Many businesses are extremely dependent on their transaction processing systems. It is important to look at every opportunity to improve system reliability when designing these systems. Such techniques as having a backup computer system or separating development activities onto a separate computer can enhance reliability. Operational systems tend to favor Oracle's transaction logging capabilities that enable the DBA to recover all transactions up to the point where a disk drive fails. Finally, when tuning these instances, it is important to emphasize resources involved with data entry, such as data and log memory areas (which are discussed later on).

The data warehouse is the second usage class to consider in this discussion. These systems are typically populated with data in overnight batch loads. They store a duplicate copy of the data so that heavy analytical queries do not impact the performance of the operational systems. Analytical queries also tend to perform better with a slightly less normalized data layout that is optimal for transaction processing systems.

Key points to consider when developing data warehouses are the distribution of computation and optimizing Oracle query resources. The distribution of computation is a balancing act related to the discussion of computer and communications architectures. It is a balancing act between having to transfer a lot of data to the PC (which can be bad on heavily-loaded networks) against having to perform a large amount of computation on the server (which is bad on heavily-loaded servers). Of course, in host-terminal configurations you do not have the option of deciding where the computation will take place. The optimization of Oracle query resources involves balancing the input/output load and resources that are often used in queries such as sort areas (this also is discussed later).

Finally, there are some Oracle instances that can be classified only as general-purpose systems. They are used for a variety of purposes. Perhaps there are only enough computer and staffing resources to support a single Oracle instance that mixes a number of functions. Perhaps the applications are tightly coupled and there is not a network infrastructure that will support multiple computers being linked together.

A fine point that becomes important to the DBA, especially in tuning, is the processing architecture of the computer running the Oracle RDBMS. Oracle in recent years has taken the approach of supporting the division of a task (query) into a number of subtasks that can be assigned to separate computer processors. Many computer vendors now support having multiple processors working together to form

a more powerful computer. You will see a number of terms thrown out, such as Symmetrical Multi-Processor (SMP, which means all processors are equally capable of getting the job done) and Massively Parallel Processors (you have a very large number, dozens or hundreds, of smaller processors getting the job done). The key to remember is that most database systems process a large number of small tasks (queries, joins, and so forth). This allows databases that can divide the work effectively to achieve high performance levels with reasonably priced processors. Oracle has been heading in this direction for several years now. It is important to understand the processor architecture on your server to know whether you will benefit from utilizing these parallel features in Oracle.

There are additional architectures out there that are beyond those discussed in this book. One is the distributed database. A distributed database is a means of storing or duplicating tables on multiple computers connected by a network. Oracle has a number of features that supports this configuration, but it is really a specialty area beyond the scope of this discussion.

So much for a whirlwind overview of environments in which you may find yourself and your Oracle database. It is important to know your environment when it comes time to make decisions such as installation, configuration, and tuning. It is time to move on to a discussion of the tools that Oracle may be interfacing with on your system.

ORACLE DEVELOPMENT TOOLS VERSUS THE DATABASE

The most basic method for access data in the Oracle database is typing SQL commands using a command interpreter such as SQL*Plus. Although this may get the job done, it is neither powerful or friendly. If you have to build local applications, you will probably want to purchase one or more development tools that fit your needs, budget, and personal tastes. This section provides a brief overview of the Oracle tools that can be used to construct applications. It will not be detailed, but it will help you understand the general concepts. When you are ready to pick tools, review recent magazine articles (products change so fast) and whenever possible, try a little hands-on to see how well it meets your individual needs.

The first tool is SQL*Plus. Recall this is not the ultimate development tool, but it does have some utility. The SQL*Plus utility is a command-line interface to the Oracle database. If you know the commands, which are a combination of SQL, PL/SQL and some special SQL*Plus additions, you can interface with Oracle. It is

not a friendly interface, but it is no worse than the dBase command line or the UNIX operating system prompt. The following are some of the uses of the SQL*Plus interface:

- ◆ Running batch PL/SQL scripts used to populate the database. This can be especially useful when you are using client-server tools for user access and do not want to acquire an expensive tool for the batch updates.
- ◆ Database maintenance. There are slightly better format controls (especially putting commas in large numbers) and the capability of editing the memory buffer that it provides. I have formed scripts for most of my routine functions (adding users and running tuning reports) that I run here, but I also issue a lot of interactive commands to troubleshoot and deal with issues as they come up.
- ◆ Testing commands to be put into larger programs so that you are sure that you have the format correct before you embed it in a trigger or large section of code that may be harder to debug.

Most mainframe applications were designed around screens (also called panels) that were displayed to the users for input and review. Because this was the way a lot of people thought, Oracle provided a screen builder that it called SQL*Forms and later renamed Oracle*Forms. It is an application development tool designed around getting information to and from the database, with structures to hold queries, updates, and so forth. The original tools were based around dumb terminals and therefore used function keys to control operations. The newer versions of this product work both with function keys and mice in a GUI environment. A key difference between Forms and SQL*Plus is that Forms is an add-on product and SQL*Plus comes as part of the cost of the database management system.

Next on our whirlwind tour of Oracle tools is the SQL*Report, which became SQL*ReportWriter, then finally became Oracle*Reports. This package enables you to form a variety of reports that have such nice features as column totals, heading, page counters, and so on. It is generally easier to make a complex report in this tool, although it is easier to get a simple list or total in SQL*Plus.

One of the difficulties these days is keeping up with the terms thrown around by the marketing types. Oracle Forms and Reports make up the bulk of the Developer 2000 tool set (you see 2000 in a lot of Oracle product names these days). There are object-oriented development tools that make up the Power Objects family. It seems funny, but there are people whose job it is to just keep track of the various products that are being introduced in the market. The pace is that quick.

Finally, a big thrust in the computer industry is the PC browser tools. Microsoft, Powersoft, and a large number of other companies have designed tools to enable you to select tables, columns, and so forth from drop-down lists in a graphical environment and use this to form queries and receive results. Oracle has made a number

of attempts at this environment, from Oracle Card to its latest browser products. It may be useful to keep this type of product in the back of your mind when you have a need for tools for people who do not want to become computer experts.

Even though the Oracle tools are separate products, many companies and agencies have agreements with Oracle to bundle these tools with the RDBMS. They are neither the best nor worst tools for application development, because it really depends on the application that you are trying to build. Just know that they may be already installed and available for you to try out (and perhaps build that one quick little report that you have been having trouble formatting properly in SQL*Plus).

ORACLE UTILITIES

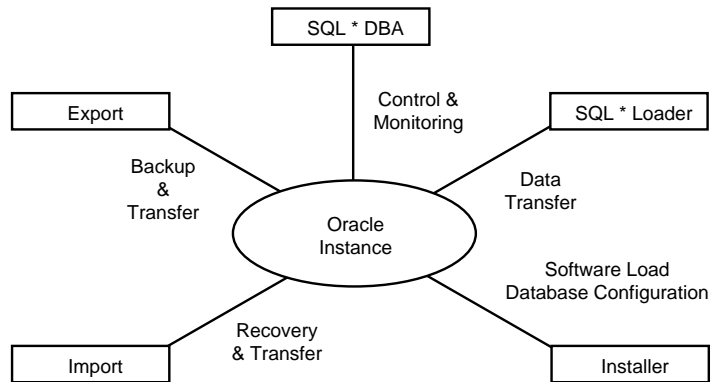
The next topic in the Oracle environment is the utilities that are included with the database. In most Oracle installations, there are four basic utilities that are used to perform support functions as a DBA (see Figure 4.2):

- ◆ *SQL*DBA*. This is a basic character—and menu-based utility that is designed to service the needs of the database administrator. Although you could design SQL*Plus scripts to perform many of the monitoring functions, certain functions, such as database startup, are always performed by SQL*DBA.
- ◆ *Export*. This is an interactive or scripted utility that enables you to take one table, multiple tables, or even the entire database to an operating system file or tape. These files can be loaded by other Oracle databases.
- ◆ *Import*. This is the companion to Oracle's Export utility. It enables you to load data from an export file into the database instance.
- ◆ *SQL*Loader*. This utility complements Oracle's Import utility. It enables you to read operating system files that have a variety of formats (semicolon-delimited, column-delimited). This is useful for transferring data from other databases or even flat-file systems. Import reads data that was stored using the Export utility. SQL*Loader is flexible enough to handle most of the rest of your data loading needs.
- ◆ *Oracle Installer*. This utility is used to install the Oracle software and build Oracle instances.
- ◆ *Oracle Terminal*. This utility allows you to ensure that Oracle works with your type of terminal—keyboard mappings, special display characteristics, and so forth.
- ◆ *Oracle Network Manager*. This utility allows you to generate the network configuration files for SQL*Net version 2. This version makes it much easier for users to specify a database to connect to; however, the burden of providing the translation between the easy names and the exact server and

instance addresses falls on the DBA. I would not recommend configuring SQL*Net version 2 without this utility.

- ◆ *SQL*Net Listener Control Utilities.* The SQL*Net protocols have special utilities (which have different names between version 1 and version 2 of SQL*Net) that allow the DBA to start, stop, and monitor the status of the SQL*Net processes. Remember—a single SQL*Net process can service many different database instances on a given host computer.
- ◆ *Special Network Tool Control Utilities* (Multiprotocol Interchange, Oracle Names, and so forth). Each of the specialized utilities that you get from Oracle seems to have its own control utility that you need to learn. Look out for these utilities and learn the control functions.

Figure 4.2.
Oracle utilities for the
DBA.

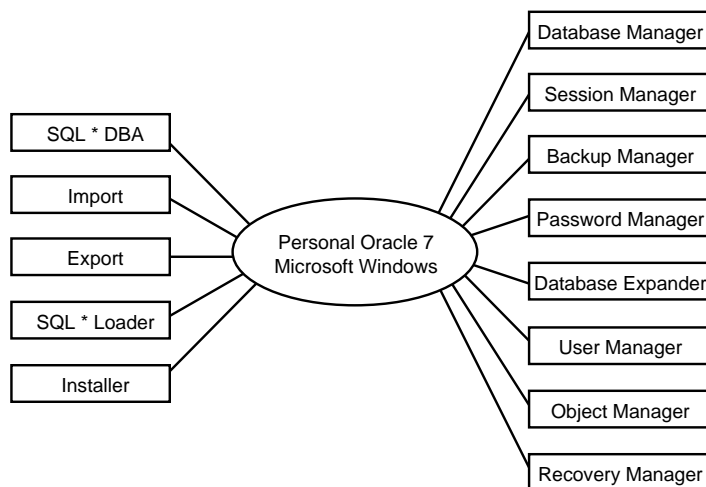


One of the interesting features of the Personal Oracle7 for Microsoft Windows product are the additional tools provided for the users (see Figure 4.3):

- ◆ *Database Manager.* This tool enables the DBA to start up, shut down, and configure initialization parameters for the Oracle instance.
- ◆ *Session Manager.* This tool enables you to monitor who is connected to the Oracle instance and kill sessions, if needed.
- ◆ *Backup Manager.* This tool enables you to perform the various forms of Oracle backup (full cold backup, partial cold backup, and so forth).
- ◆ *Password Manager.* This tool enables you to change the database password. Because Personal Oracle runs under Microsoft Windows, which lacks password control, Oracle has implemented a special password that is used to access key database utilities, such as backup manager, user manager, and so forth.
- ◆ *Database Expander.* This tool enables you to add space to the various tablespaces by adding data files.

- ◆ *User Manager*. This tool enables you to perform the routine user administrative tasks, such as adding users, changing roles, setting system privileges, and so forth.
- ◆ *Object Manager*. This tool enables you to maintain database objects such as tables, indexes, and so forth. Typical functions include creating, deleting, and setting privileges for the objects.
- ◆ *Recovery Manager*. A complement to the backup manager, this tool enables you to perform database recovery functions.

Figure 4.3.
Additional tools in the
Personal Oracle7
server.



Whichever environment you work in, the basic tools described in this section enable the DBA to get the job done. Many DBAs would like to see improvements in the tools, especially in the areas of friendly interfaces and, in some cases such as the installer, fewer bugs. Oracle has put special effort into the Personal Oracle7 tool set to provide a number of functional and friendly GUI-based tools that enable you to point and click your way through database maintenance. In addition, there is the Oracle Server Manager product designed for larger Oracle systems that provides a better tool set to maintain the databases (if you have a graphical environment from which you can access all your instances). Finally, with the large number of Oracle instances out there, there are a number of third-party products designed to meet some of the needs of the database administrator. The next section presents some of the features of these third-party tools.

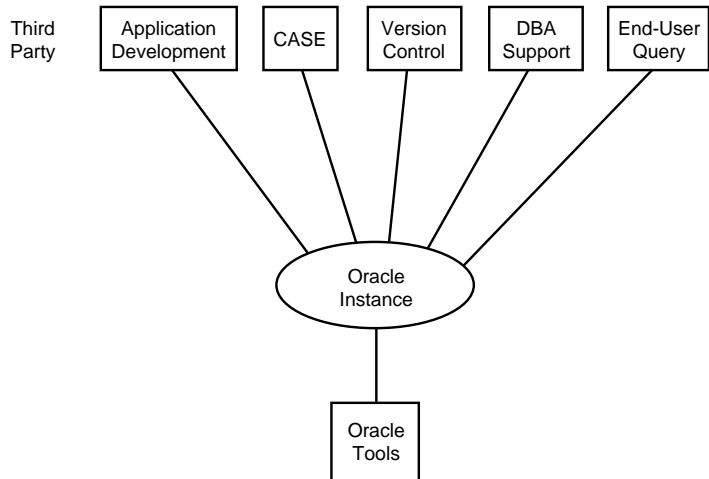
THIRD-PARTY PRODUCTS

This section explores some of the third-party products that interface with the Oracle environment. By the time this book is published, each of these products will have gone through one or more upgrades that will strongly effect its suitability towards your needs. The best place to review the functionality of tools for your environment is the computer magazines. However, I strongly recommend that you also talk with other people who are using the tools and, whenever possible, get some hands-on experience (demonstrations, evaluation copies, working with someone else's system for a day, and so forth) before you commit to a particular product.

With those caveats out of the way, it is time to move on to a discussion of these products. Much as with the Oracle products, the third-party vendors offer many different types of products that can fit in with the Oracle environment (see Figure 4.4):

- ◆ Application development tools
- ◆ Computer-Aided Software Engineering (CASE) tools
- ◆ Version control software
- ◆ DBA support tools
- ◆ End-user query tools

Figure 4.4.
Common third-party
products in an Oracle
environment.



There are a number of different application development tools for the Oracle database. Oracle sells a large number of databases; therefore, there is incentive for a number of companies to develop and interface application development tools with the Oracle database environment. It can actually be a problem to sort through the

large number of competitors who are trying to sell you their products, but this is better than having no choices at all. These application development tools tend to fall into the following categories:

- ◆ Report-building tools used to generate a variety of report formats, similar to Oracle Reports.
- ◆ Screen builders used to generate data entry and viewing screens similar to Oracle Forms.
- ◆ Programming languages (C, COBOL, FORTRAN, and so forth) that can be interfaced with the Oracle database through the use of pre-compilers. These pre-compilers are used similarly to subroutine libraries to turn SQL queries into the appropriate format to transmit queries to and receive data from the Oracle instance.
- ◆ All-encompassing development tools such as PowerBuilder and Gupta. These usually combine screen building, report generation, programming language constructs, GUI interface generation, and even some graphics generation capabilities.
- ◆ Decision support development tools. There are a number of tools designed to enable developers to formulate the what-if type of queries rapidly and to enable users to drill down through multiple layers of detail rapidly. These tools are somewhat specialized and have a smaller market share than the other tools mentioned.

Typical decision points to determine which tools meet your needs include the following:

- ◆ What platform is available to the users? Most of the products are limited to one environment (UNIX host computers with terminal emulation or MS Windows via client-server interfaces).
- ◆ What requirements do the users have for ease of use? For example, what is needed: a GUI interface, a menu-driven interface, or a spreadsheet-like interface?
- ◆ What is the background of your existing development staff? Retraining in new tools can add up to substantial costs.
- ◆ What is your budget? Some of these tools can be quite expensive and others are relatively inexpensive.
- ◆ Can the tool handle the size loads that you need (from small applications and development groups to larger ones)?
- ◆ What data architectures does the tool support (host-based, client-server, three-tier client-server, and so forth)?
- ◆ How much in the way of resources will be required for the tools and can they provide the needed performance enhancements?

- ◆ Are you entering a highly proprietary environment or an open one? Can these tools be integrated easily into your current environment?

Most of the computer shops that I have encountered in my travels have difficulty keeping up with the changing application demands of the user community. It is usually much easier to think up a new report than it is to build it. This often results in a large application backlog, disgruntled users, and overworked programmers. Some vendors have developed what I call end user query tools (there are a number of words floating out there in the industry including EIS, flexible query tools, and data browsers). These tools all have similar goals—to enable users to get at complex data stored in the database without having to understand networks, communications, or even the SQL language. Different vendors approach this problem from various perspectives. Some try to decode normal English questions and turn it into SQL queries. Others emphasize letting you pick tables and fields from a GUI display to produce your results. Finally—one of my favorites—is a spreadsheet. I once had a user who could never get through what he wanted to any of his programmers. He had no desire to give up engineering to learn programming but he, being a manager, knew how to use spreadsheets. We connected his Excel spreadsheet to the Oracle database and he got more done in a few days than was accomplished in all the weeks prior. Never overlook the simple solutions to complex problems.

Computer-aided software engineering (CASE) is another field that has a number of third-party vendors. I was told few years ago that CASE was *the* solution to all problems with application development. I don't think this revolution has come about, but there are many circumstances where a CASE tool can come in handy. Many CASE vendors offer integrated suites, which automatically generate software or capture the logic of existing applications for re-engineering efforts. Many of these vendors offer CASE products as their primary line of business. If, as a DBA, you see developers wandering or realize that they do not understand relational database design concepts, you may suggest that they look at CASE tools. This will not replace basic knowledge of database design, but if you help them a little, the analytical tools can point out many flaws in their database design. You could also consider one for yourself if you become responsible for ensuring that there is no duplication of data in your Oracle database among multiple applications, especially if you have a large, integrated corporate data architecture with a number of developers and development projects.

An interesting set of development products out there are those products designed to control the version levels of software applications developed. There are some that can even control versions of the data definition language that you use to create database objects, such as tables and indexes. These packages support different software development tools. Some software development tools also come with

version control features included. Keep these in the back of your mind if you also have system responsibilities and your developers are known for overwriting production software incorrectly.

Finally, there are a number of tools out there designed to support the database administrator. There are tools that range from a neat series of SQL scripts to automate a few of the more tiresome DBA tasks to complex applications that can display every parameter that can be monitored by the Oracle database. Some even send you mail when it finds a problem with your system. The trade journals contain ads from these vendors. They also often demonstrate their products to local users' groups. It is a good idea to know what these tools are in case you need help on your Oracle instances.

ASSEMBLING THE PARTS INTO AN ARCHITECTURE

So far in this chapter, you have explored an environment where there are a large number of choices for each component in your database and application architecture. There are some in the computer industry who long for the day when they bought everything from IBM, developed applications, set their organizational structure based on their guidance, and paid whatever they asked. For the most part, I believe that era is long behind us. Even IBM offers four major lines of computers (PC, UNIX, AS/400, and mainframes) with a number of operating systems, database choices, development tools, and communications architectures. Therefore, almost all data centers are forced to provide some level of integration of components to complete their overall architecture.

This is actually one of my favorite topics. I have done a number of major (multimillion-dollar) system designs and acquisitions and have also come in as a consultant for a number of systems that were about to be installed but were missing significant components. Most DBAs are not in charge of system acquisitions. The focus of this chapter is to arm you with some knowledge. Using this knowledge, you should be able to provide useful input to the acquisition process so that you get all the parts that you need to get your database working.

Every type of system that you acquire has its own special problem areas. The Oracle database is no exception. Perhaps the word "problem" is too severe. Every application needs certain resources to perform its tasks. Engineering analysis software tends to stress floating-point computational capabilities of the central processor. Oracle tends to stress everything else—integer processing capabilities, input/output, and memory storage capabilities. Based on the recommendations in the

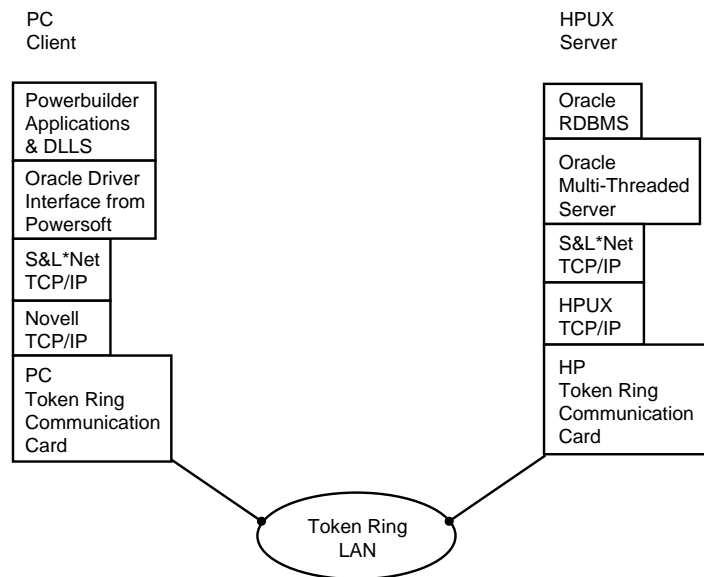
Oracle Installation and Configuration Guide and my experience, here are a few recommendations when you are planning out an Oracle installation:

- ◆ The minimum memory recommendations for Oracle are just that—minimums. If you anticipate having a large number of users or will be performing large sorts in memory, you should consider having additional memory available. Unfortunately, it is very difficult to calculate this in advance. The Oracle Installation and Configuration Guide lists memory requirements for products, but it cannot anticipate the size of your sorts or the update volume of your users. Ask your vendors to provide a list of other customers using their products for a similar project and find out how much memory they are using.
- ◆ The next requirement is input/output capability. Having a number of different controllers and disks to split the input/output tasks is generally a good idea. Following Oracle's recommendations on splitting tables, indexes, temporary space, and log files is beneficial to most instances that perform a high volumes of queries and transactions. Of course, if you have a small instance, it can fit on one disk. Again, looking at similar projects should provide a clue as to what you should be doing.
- ◆ Read the version requirements for Oracle products before completing the system design. Unfortunately, the first time most DBAs get to see these requirements is when Oracle and its Installation and Configuration Guide arrive. Although this is usually not a problem on new systems, older systems may need to have certain components (the operating system release level) upgraded before Oracle will work reliably. This is when you should call upon the salesman's technical support staff to provide you with this information when forming the order (and before the budget is completely spent).
- ◆ Be extremely careful when implementing the trusted version of Oracle or the operating system. Most commercial applications (accounting packages that work with Oracle) do not work with the trusted version of the database.

The host-based architecture is perhaps the easiest to implement from a DBA perspective. The Oracle products are loaded onto the host computer. There is no SQL*Net component to set up and monitor, unless you want to take advantage of the multi-threaded server architecture. (You learn more about the multi-threaded server later.) The applications have the relatively simple interface directly to the Oracle processes on the same computer. If you have been careful about checking version requirements and have ensured that all vendors have listed their detailed requirements for other products, you should be able to implement this environment with relative ease. Be prepared, though. You may have to work out some bugs with installation, bad media, and other problems. If this is your first installation, leave yourself plenty of time.

Client-server architectures provide a lot of functionality to the user interface but require a little extra work from those installing the architecture (that is, you, the DBA). For example, Figure 4.5 shows a sample installation performed with applications built using Powerbuilder (a leading PC GUI forms-and-report application development tool) on PCs interfacing to an Oracle database on an HP UNIX box. The customer decided to use TCP/IP as the network protocol to communicate with the UNIX boxes (most UNIX boxes treat TCP/IP as their native protocol). The Novell TCP/IP communications products were used. This customer had to deal with vendor representatives saying that they interfaced with the others' products, but not mentioning that the add-on product that had to be purchased from that vendor to make it work. They also did not know that when you buy SQL*Net, you have to buy separate products for both the server and the client computers. It's a good idea when dealing with vendors you have not worked with before to make sure that they put in writing what all computers and network components need to have in order to make their product work, down to the version level. Then you have it in writing later on when you find problems.

Figure 4.5.
Sample client-server
components.



The network infrastructure is the next component that can cause problems when integrating an Oracle database. It is usually impossible for the average end users to know whether it took so long for their report to run because the database was slow, the application was poorly designed, the host computer was overloaded, or the network capacity was inadequate. The DBA is the first person blamed in most places, so it is useful to ensure that the network bandwidth is adequate, from your

users to the database computer. Many locations transmit every network packet to every computer in the building, which overloads the network. Others have nice local area networks, but smaller wide area network paths that serve as communications bottlenecks. It is not your job to design the network. However, you should recommend that network capacity is considered. Network performance problems are especially common in locations that are just beginning to use networks for more than PC print-server communications.

A technique that is always useful when evaluating products is the evaluation copy. Many vendors will loan you a copy of their software for a specified period so that you can try it in your environment. That way, you know whether the product works with your network and operating system release level. It can also be useful to enable end users to participate in the evaluation process so that they get a feeling that they are part of the solution. Once I was doing a documentation publication system where I put in a number of UNIX and PC workstations with various monitors and graphics cards for evaluation. I was very surprised when the users picked a smaller monitor as their favorite in this evaluation. Apparently, when you look at these large screens all day, sharpness and freedom from flicker are more important than size. You may find the same is true with the look and feel of database development tools.

For larger orders, you can sometimes get the vendor to set up a full-scale demo of what you would be implementing. Many hardware vendors have relationships with software vendors such as Oracle. They can set up a demonstration copy of Oracle on a computer similar to the one that you are considering purchasing in their offices. In that manner, you get to see how it would actually look and respond if you do not have the equipment in-house to perform evaluations.

Another thing to do on larger integration efforts is contacting other customers of the vendors. It is especially useful if you can find people who are doing a similar application with similar products that will have a database that is about the size of what you are planning. Most of the ones that you contact will be willing to talk at least briefly over the telephone about what they bought and why. When you combine several of these contacts, you can usually flesh out a number of problems with the various products. You can then press the vendors for solutions, which are often merely an additional product that has to be included in the order. It is often not the list of products that determines the performance, but the size of the database that can make or break performance (that is, you get lookup tables that are just large enough not to fit in memory for sorts).

Do not assume that your system will stay static after installation. Although there may be a few places that can guarantee that the applications, data, and processing needs will not grow, I have not found any of them. I usually see a rush to build more

applications once the first one is developed, especially when applications move from overloaded mainframes to smaller client-server platforms with new development tools. Many users never realized that they could issue a specific query to find a list of all orders placed by a particular client instead of sorting through hundreds of pages of printouts. It's a good idea to purchase a little bit of extra capacity up front. I also tend to favor purchasing systems that can be upgraded with relative ease (able to add processors, disk drives, expansion cards, and so forth). In some environments this is not allowed, but if you can factor it into the design, it can save a lot of headaches later on.

System integrators are organizations that have the advantage of having done what you are trying to do. They should be aware of the products that are on the market and able to show you alternatives to meet your needs. A few words of caution:

- ◆ Always be careful about integrators who work with only one or two product lines. There may be products out there that would be better for your needs that you would not get to see with this integrator.
- ◆ Ensure that you know what you want functionally before you sign a contract. Too often people get what they ordered only to realize that it was not what they wanted. Whose fault is it? Ensure that you can articulate what you want in terms of performance, number of users, amount of disk storage, and so forth. Going back to the expandability topic, ensure that you put requirements in there so that the system can be expanded to a certain size without having to dispose of major components (such as UNIX servers that have all their disk bays, expansion card ports, and memory slots filled when initially installed).

Building an environment for the Oracle database could be the subject of an entire book, but this is good enough for now. It may not be part of your job, but it could keep you from doing your job if something is missing. This is especially tough in places that are shifting to a radically new environment (mainframe to UNIX client-server, for example).

SUMMARY

This chapter presented an overview of the various environments in which you can find an Oracle database. It is not as important to focus on the process configuration differences between IBM mainframe and HP UNIX versions of Oracle, but the whole architecture has a number of variables that you have to understand. Of course, in addition to understanding, it is useful if you can provide input to the acquisition process to ensure that the components that arrive on your doorstep just before the system is due to become operational will actually work together.

You will probably not use the material in these introductory chapters in your day-to-day work. However, it is important to lay the foundation of your understanding of the job and the environment for the later chapters. Chapter 5 discusses topics that you will deal with almost every day and presents a more detailed discussion of the DBA tools introduced in this chapter.

After the introductions in the last several chapters on the database administrator's job and the environments of Oracle, it is appropriate to provide you with an introduction to the tools that you will be using to perform your work. As with any other toolkit, it is important to know what the various tools are, what they do best, and how to use them. That knowledge is the goal of this chapter. It takes actual practice using the tools to become proficient, but this should get you started.

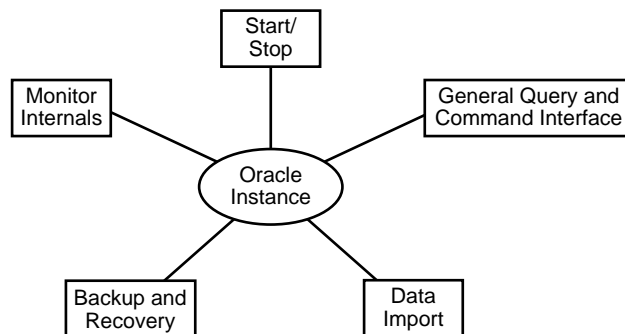
You will not necessarily use every tool. Some locations never use exports of the database for backups or data extent compression. Other locations may never need SQL*Loader to move data from another system into their Oracle instances. The sections on Personal Oracle7 are applicable to only that one product. Feel free to read only the sections that are applicable to your job needs. However, it never hurts to have an understanding of what else is available, in case your needs change.

Before going through the list of tools provided by Oracle, let's look at the conceptual tools that you need to perform as a DBA. After that, you'll map the Oracle tool set to these requirements. This will not include discussions of development tools, configuration management utilities, and so forth. You may use some of these tools to build reports for your use and other tasks, but they are not part of the core set of tools you need to get the basic job done.

There are many ways to categorize the types of tools that a DBA may need. For purposes of this discussion, let's use the five categories shown in Figure 5.1:

- ◆ Starting and stopping the instance
- ◆ Monitoring the internal workings of the instance
- ◆ Backup and recovery
- ◆ Importing data from other systems
- ◆ General query and command interface

Figure 5.1.
Tools needed by the DBA.



The first conceptual tool needed is one to start and stop the instances. This control needs to be on an instance-by-instance basis, because you may need that on computers that have more than one Oracle instance. As you will see later, you also may need some control over how far to start the instance. This may include starting just the processes, mounting the database, and opening the instance for use. For shutdowns, you may want to wait for everyone to disconnect from the system before the shutdown occurs, shut down whether they have disconnected or not, or perform an emergency shutdown when there are problems that prevent a normal shutdown.

The next conceptual tool for the DBA enables the DBA to monitor the internal workings of the Oracle instance. Let's hope you will rarely run into situations where you need to check the status of these internal parameters. However, when the system is generally running slowly or a particular user is locked up, you will want to access this utility. Such a utility should enable you to see all who are logged in, what they are doing, and what the Oracle background processes are doing. It should show you the status of all of the Oracle files and internal mechanisms, such as table locks.

An important set of DBA tools support the backup of the database and recovery of the data in case a data file is damaged. This utility should enable you to get all the data in the database, in a tablespace, or in a set of data tables. It should be able to capture information related to a table, such as indexes and grants. Finally, it should enable you to build scripts that will perform automatic backups of the database.

Many Oracle instances need to be able to import large quantities of data from other systems. The classic example of such a system is a data warehouse that receives almost all its information from an operational transaction processing system. It could be debated whether this task belongs to the programmers, to operations staff, or to the DBAs. Some places run all production jobs through the operations staff, but I have seen other places where the DBA has to do it. It is included under the DBA category here because these loads are often so large that the DBA is assigned the task to keep an eye on space. The basic requirement for this service is to import text files that are either delimited by columns, such as flat files, or delimited by a specified character, such as a comma.

Finally, a general-purpose query-and-command interface is needed to support most of the other DBA tasks that come up. The function is simple—to enable DBAs to issue SQL queries to the DBA tables and views that contain information about the configuration and performance of the system. A side requirement is to enable the DBA to format the output somewhat and spool the results to a text file or to a printer.

Now that you know what is needed by the DBA, it is time to discuss how these needs are met by the Oracle products that you will have available to you when you purchase Oracle. There are third-party add-on products that may perform one or all

of these functions. However, the utilities discussed here are shipped with the RDBMS software, so you can count on having them when you start up your system. This section presents two categories of utilities—those under most multi-user operating system versions of Oracle (UNIX and VMS) and those under Personal Oracle7.

Under most multi-user versions of Oracle, you rely on a series of five utilities. SQL*DBA performs all of the functions described previously, with the exception of loading data from other computer systems. SQL*Plus is a command-line interface that can be used to support general queries, backup, and recovery and also certain monitoring functions. SQL*Loader provides services for loading data from other computer systems. Oracle Import and Export are designed to perform backup and recovery services. Together, these utilities provide the basic services needed by the DBA. Although some of them may not be very pretty and you may not like the way they are designed, they will get the job done for you. Each of these utilities has a chapter or more in the utilities user's guide that comes with Oracle.

THE BASICS: SQL*DBA

The first utility is the one originally designed with the DBA in mind. SQL*DBA is the utility that I use for starting up, shutting down, and using various monitoring utilities when the Oracle instance is not performing properly. Like most tools, it has a series of pros and cons to consider:

- ◆ First, it connects you to whatever Oracle instance is specified by your operating system parameters (ORACLE_SID and ORACLE_HOME). You cannot easily switch between multiple instances on a single server or to other instances that you are responsible for on different servers unless you have SQL*Net installed. If you have SQL*Net, you can use the `connect username@connect-string` format to connect to these other instances.
- ◆ It is a text-based environment with pull-down menus. This works well in environments where the DBA connects to the server with a simple terminal emulator. However, many of us have become spoiled and long for the convenient and powerful features of a graphical user interface (buttons to click for commands and bar charts to display results).
- ◆ It provides monitors for most of the parameters that you need to check for routine database problems (lock status and disk input/output status). You can issue SQL queries from the command line to look at parameters and data that are tracked by Oracle but not displayed on one of the menu outputs.

- ◆ Perhaps the most important feature of all for new DBAs is that SQL*DBA is provided with the Oracle RDBMS software. The version of SQL*DBA is designed to work specifically with the version of the Oracle RDBMS that you have purchased. If you choose to purchase third-party DBA support tools, you may encounter problems when you upgrade to a new version of Oracle. SQL*DBA will be there for you to fall back upon.

SQL*DBA functions in two different modes. The first mode is the command-line interface where you type SQL-like commands to get a particular job done. If you are having terminal emulation problems or wish to run scripts from utilities such as `cron` (the UNIX automated job scheduler), you use the command-line mode. The other mode presents a pull-down menu utility that enables you to select the task you wish to perform. Figure 5.2 presents a sample of this menu taken from an Oracle 7.1 instance under UNIX. An interesting note is that many of the functions that you would use from the menu mode in most other installations are not available in SQL*DBA for the Personal Oracle7 Server. Instead, there are separate icons within your Oracle program group that provide true GUI tools to perform these functions. At first, I groused at the idea of having to use a separate utility to expand the database or monitor user sessions. However, after trying them, I began to like them. The CPU time required to use convenient GUI tools may be too much for someone trying to use an old 20-MHz PC to run Oracle, but on my Pentium P60, it was worth it.

Figure 5.2.
A sample SQL*DBA 7.1
menu under UNIX.



There are probably enough topics on SQL*DBA to write a small book on the subject, but there's not room for that here. Instead, study Table 5.1, which presents the SQL*DBA functions that you would find in an Oracle 7.1 instance under UNIX. With experience, you will get comfortable navigating the menus or entering the

commands needed to actually execute the commands. Recall that Personal Oracle7 provides separate GUI-based utilities to perform most of these functions and does not put these on its SQL*DBA menu. Earlier versions of Oracle may provide fewer utilities, so if you cannot find it on your menu, you have to try the SQL queries to the database tables. Most of these commands are pretty clear as to what they do and the activities that they perform; they will become clearer when you understand the tasks that you will be performing. (See the monitoring discussion in Chapter 30 for some of the more common queries that you need.)

TABLE 5.1. FUNCTIONS PROVIDED BY SQL*DBA.

<i>Type of Function</i>	<i>Functions Available</i>
File	Run SQL script
	Spool on
	Spool off
	Quit
Edit	Cut/paste/copy/clear
	Previous/next command
Session	Connect
	Disconnect
	Enter system command
Instance	Start up
	Shut down
	Mount database
	Open database
	Force checkpoint
	Force log switch
	Configure dispatcher
	Configure shared server
	Prevent/allow connection
	Kill session
Storage	Tablespace
	Rollback segment
Log	Add/drop group
	Add/drop/rename member
	Enable/disable thread

<i>Type of Function</i>	<i>Functions Available</i>
Backup	Start/stop auto archive
	Begin manual archive
	List archive status
	Begin/end online tablespace backup
Security	Recover database/tablespace/data file
	Create/alter/drop user
	Create/alter/drop role
	Create/alter/drop profile
Monitor	Alter resource cost
	Grant/revoke
	Multi-threaded
	Process
	Session
	Table
	SQL area
	Library cache
	Latch
	Lock
	File I/O
	System I/O
	Rollback
	Statistics
Help	About SQL*DBA
	Help system
	Show keys

THE NEXT GENERATION : ORACLE SERVER MANAGER

Oracle intends the Oracle Server Manager to be a successor to SQL*DBA. To be honest, I had to learn the DBA job before the menus in SQL*DBA were as convenient as they are today, so I issue most of my DBA commands from the SQL*DBA

command line. How barbaric and primitive you might say, but for me it is quicker. Even though I have experimented with Oracle Server Manager and read about it, I have not used it much. For one thing, the version that I worked with was designed for Sun's X Window display environment. I have run across a number of X Window displays and terminal emulation packages in the scientific community, but only one that I can think of in the business world. There was only one user who had PC X Window connections and it was somewhat slow.

The key consideration for Server Manager is that it is designed to provide a very friendly graphical user interface environment (you get to use your mouse and buttons on the screen versus pull-down menus) for you to do the things that you did in SQL*DBA. Functionally, you can think of it as performing the functions of SQL*DBA, with a much better interface. Another good feature is that it enables you to connect to any of the database instances that are accessible to you through the network. In this fashion, you can maintain all your instances from one desktop. I have maintained five instances at one time located on three different computers. I can testify that one window that enabled me to connect to any of these instances would have saved a lot of time and confusion logging into the various computers, remembering which was my default instance, and so forth.

PERSONAL ORACLE 7 FOR MICROSOFT WINDOWS

When designing a software system that works on multiple computer platforms, developers are faced with a dilemma. Users who are familiar with your products in other environments want to see the same thing when they come to the new environment. On the other hand, users who are new to your product, but experts in the environment, want to see you follow all the standards and conventions that apply to that platform. When Oracle moved into the Microsoft Windows environment in early 1995, they came up with a good compromise between the two worlds.

Oracle kept the five basic utilities described previously. SQL*DBA lacks most of the pull-down menu picks described earlier in this section. SQL*Loader, Import, and Export have graphical user interfaces added to make it a little easier to deal with than the question-and-answer interfaces. SQL*Plus is basically the same in both environments, although you have access to things such as the clipboard and other Windows features under the Personal Oracle7 product.

Where Personal Oracle7 departs from the rest of the Oracle family in a pleasant manner is the new GUI-based tools to help the database administrator. These tools were necessary in the PC world wherein developers and users may be asked to perform many of the database administrator functions on their databases. Most of

these tools have the title “Manager,” but don’t let that fool you. They actually do useful work. Let’s start with a list of the tools and then go into their functions:

- ◆ Database Manager
- ◆ Session Manager
- ◆ Backup Manager
- ◆ Password Manager
- ◆ ODBC Administrator
- ◆ Database Expander
- ◆ User Manager
- ◆ Object Manager
- ◆ Recovery Manager

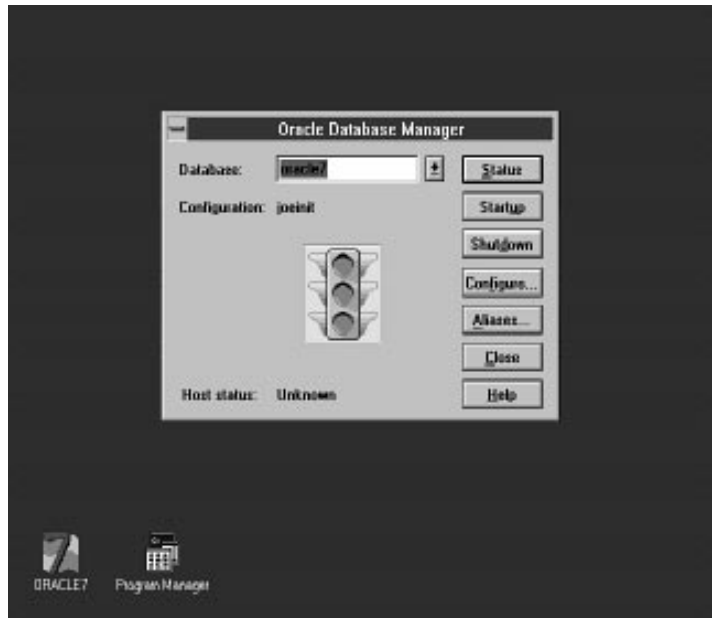
Database Manager is one of the first products that you will encounter when you load Personal Oracle7. One of the best features is that you are prompted to see what instance you want to work with when you connect to the tool. Another good feature is the status option, which enables you to determine whether the instance is up or not. In most UNIX systems, you have to get a listing of the processes and sift through it looking for the Oracle processes associated with a given instance to see whether the instance is up. Finally, a feature that is really convenient is the configuration windows that are available from this tool. You can reconfigure your Oracle instance without worrying about editing the init files. Of course, many DBAs who have had to use the infamous vi editor in UNIX to edit init files have little sympathy for people who think notepad is too hard in Windows. Figure 5.3 shows the main panel from Database Manager.

The next Personal Oracle7 tool is Session Manager, which combines the session monitoring and killing capabilities of SQL*DBA in a convenient user interface. Occasionally, you get a user who issues a particularly long query that cannot be stopped for whatever reason. The DBA is usually called in to kill this process. A query process sometimes become detached so that the user cannot kill it. (I have never seen this under Personal Oracle7, but the utilities are here and Session Manager is the tool.) Figure 5.4 shows the simple interface provided by session manager.

Backup Manager is the next tool in your toolkit. It is a simple utility that does what it advertises. It also enforces some rules that are probably good for the general public, but that may frustrate some Oracle DBAs. One rule says that if you are not operating in archivelog mode, you have to do a full cold backup (you cannot use the other options). Because I do not have a tape drive on my PC, I like to have the option of backing up only tablespaces that contain persistent data and not backing up the

temp tablespace. If I lose the disk, I merely drop and re-create those tablespaces. I guess that I am spoiled by the Microsoft Backup utility that enables me to write data to multiple disk drives. I am pretty religious about backups of even my home system. I know I can make a backup of the database to a disk file and then use Microsoft Backup to copy these files to disk, but this is one suggestion that I would put in the queue for Personal Oracle7.

Figure 5.3.
Database Manager.



The next utility is unique to Personal Oracle7 and deals with extra security requirements needed under Microsoft Windows. In most implementations, Oracle relies upon operating system password security to mark who is a DBA and can perform functions such as database start ups. Because Microsoft Windows does not require passwords (although some hardware passwords and screensaver passwords can be used), Oracle added an additional layer of security for these sensitive database functions. You have all your personal Oracle user IDs, such as sys and system, but there is also a special database password that is not controlled by the normal SQL commands. This utility is where you can reset this database password.

The ODBC Administrator (open database connectivity) deals with a special feature of Personal Oracle7 that is designed to enable it to interface with data files from other popular PC databases. ODBC refers to a standard developed by a number of vendors, primarily from the PC arena, that enables databases to exchange information. Figure 5.5 shows the relatively simple interface and some of the default connections that can be provided by this utility.

Figure 5.4.
Session Manager.

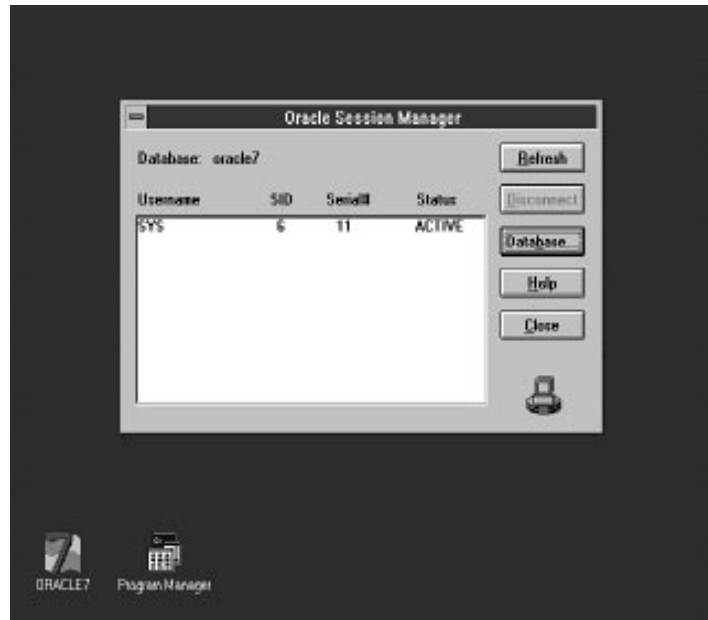
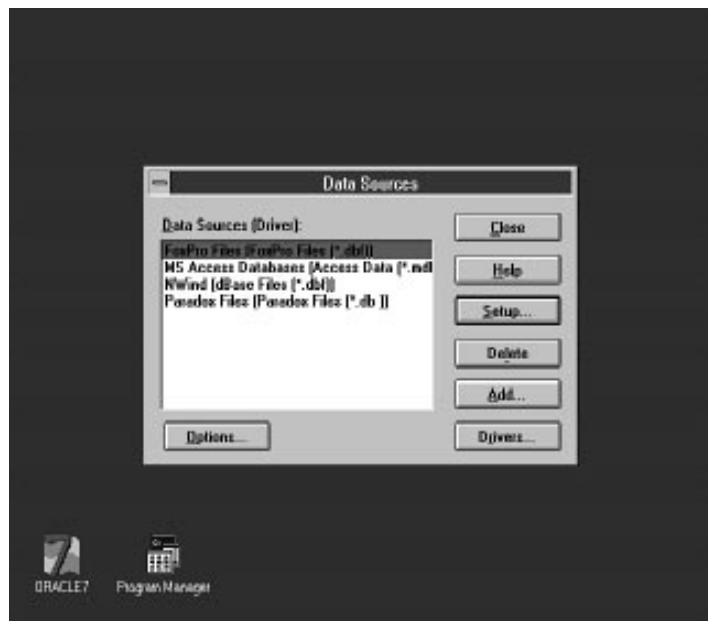


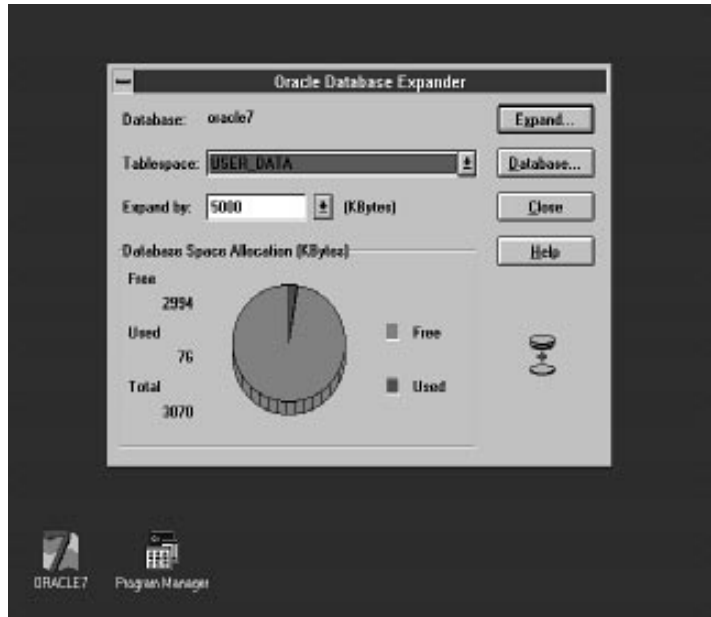
Figure 5.5.
ODBC Administrator.



For DBAs experienced with the UNIX world or who wish precise control over where their data is placed, SQL*DBA and SQL*Plus are fine tools to issue the commands to add data files to a tablespace. However, the Database Expander is a good tool for

those who do not want to go to this bother. This is especially true in the PC environment wherein you usually have a single disk drive, so the ability to specify the exact location for data files is not as important. Oracle has thrown in a few nice features, such as a drop-down list box to enable you to select the tablespace and a graphic showing how much of the existing tablespace is used (see Figure 5.6).

Figure 5.6.
Database Expander.



The next tool in this tool set is the User Manager. This is a little different from some of the other tools in that you are presented with the list of users and a number of buttons that represent functions that you can perform for those users (see Figure 5.7). You click on a user and select the button for the function. You then are presented with another window in which you enter the new password, select the desired privileges, and so forth. This is very convenient in that it saves you from having to remember the exact name of the roles that you created or having to retype long SQL commands because you misspelled something.

The Object Manager enables you to create or select objects in the database and modify such things as privileges or structure of the object (see Figure 5.8). It can turn out to be a very convenient tool for those who have not yet learned to love the SQL language and command prompts.

The final DBA tool in the Personal Oracle7 family is the Recovery Manager. Perhaps what I like most about this tool is that it is simple and provides you with a lot of radio buttons at a time when you are probably a little stressed (see Figure 5.9). It does not provide you with many fancy options, which is probably not as important given the

relatively simple structures of most Personal Oracle7 databases. I have not done extensive testing on this product, but it did the job when I tested a simple complete recovery.

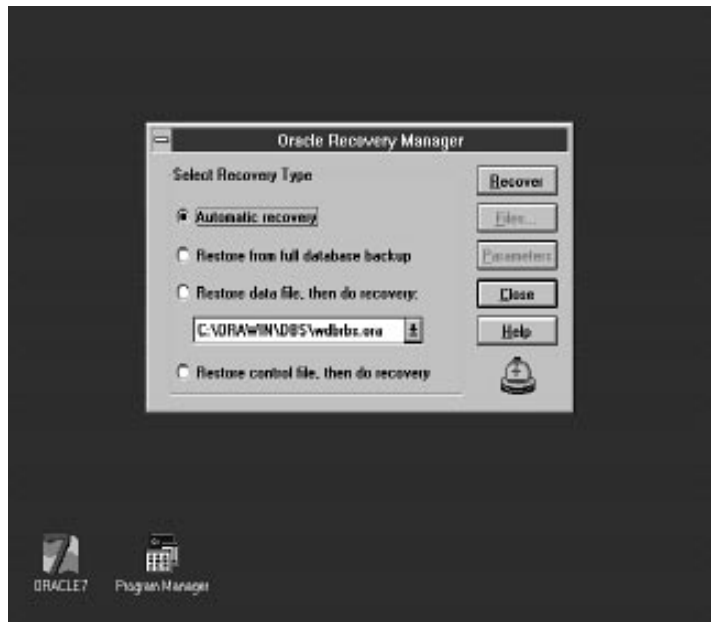
Figure 5.7.
User Manager.



Figure 5.8.
Object Manager.



Figure 5.9.
Recovery Manager.



These are the DBA tools in the Personal Oracle7 environment. I have to admit that I like them much better than the tools that I am used to in the UNIX Oracle world. However, when I am in a hurry, I still tend to open an SQL*Plus or SQL*DBA window and issue the SQL commands manually. Perhaps I am not comfortable giving up control of the exact placement of files. Perhaps I am on my way to becoming a dinosaur who sits around and complains about how real DBAs should do everything from the command line. Or perhaps I will get used to them over time and wonder how I ever managed to get things done from those command lines.

THE COMMAND-LINE INTERFACE: SQL*PLUS

SQL*Plus is the basic command-line interface to the Oracle database. It does not have a pull-down menu structure that enables you to pick common DBA commands. It does however provide a slightly more sophisticated formatting and script handling capability. I tend to use SQL*DBA for monitoring and startup/shutdown functions and work most of the other tasks with SQL*Plus. Perhaps it is just that I get fussy about wanting my output to have commas between the numbers and other such features.

The basic function of SQL*Plus enables you to enter and run SQL and PL/SQL commands to interact with the database. There is not the space here to provide an introduction to the wonders of the Structured Query Language. (A summary of some

of the more common commands is presented in Appendix A.) The following are a few of the features of SQL*Plus that can be used to improve your output:

- ◆ One of my favorites is the page length and pause options. Basically, I set page length to be 24 (24 lines of output before a page break) and then set pause to on (stop and wait for a return after every page). That way I get to view a large amount of output one screen at a time. Here are the commands to implement this:

```
set pagesize 24;
set pause on;
```

- ◆ Another favorite of mine is the capability of specifying the output format of columns. This enables you to specify the width and also perform formatting, such as putting commas between thousands in numbers. The width feature is important when accessing a number of DBA views that have very large columns, but which are not commonly filled to their full size. For example, the owner column is typically 30 characters, but because most installations use shorter Oracle IDs, you can reduce the display width to 10 or 20 characters when you are trying to get a number of columns to fit across the page. The following example shows some of what this SQL*Plus formatting can do:

```
SQL> column bytes format 999,999,999
SQL> column tablespace_name format a15
SQL> select tablespace_name,bytes from dba_data_files;
```

TABSPACE_NAME	BYTES
ROLLBACK_DATA	3,145,728
TEMPORARY_DATA	2,097,152
USER_DATA	3,145,728
SYSTEM	10,485,760

```
SQL>
```

- ◆ A useful feature on systems where you cannot print or capture output buffers freely is the spooling feature. This enables you to redirect to a file or to a printer. The following is a sample format for this command:

```
spool output.txt
```

- ◆ Another feature that I often use in SQL*Plus is the capability of loading script files that have been previously saved to disk, modifying them, running them, and then saving the modified files back to disk. The command to retrieve a file is `get`. The command to modify the contents of the buffer is `edit`. The command to save the contents of the buffer to a disk file is `save`.
- ◆ A useful design decision Oracle made was to integrate with the host computer editing tools. If you are used to the `vi` editor, you do not have to learn a new editor just to work in Oracle.

- ♦ A design feature that saves time offers the capability of executing operating system commands from within SQL*Plus.

This has been a brief overview of SQL*Plus. If you still have questions about this utility, Oracle has a book devoted to the subject that comes with the basic system documentation.

IMPORT AND EXPORT

Oracle's Import and Export utilities are two of the backup options available to the DBA. These utilities are useful in reconfiguring database objects (for example, resizing tables, which is explained later) and transferring tables between Oracle instances. The advantage of these utilities over SQL*Loader (which is described in the next section) is that when they export a table, they capture not only the data but the table structure, storage parameters, grants, indexes, and so forth. Unlike the other Oracle backup mechanisms, Export also enables you to select a specific list of tables rather than having to back up an entire tablespace.

Export is a relatively simple command-line utility to work with. It either prompts you for the operational parameters it needs to perform the export or you can pass these parameters in on the command line. The following example shows the basic parameters in an interactive Export session. All these parameters can be passed in with a simple syntax of *parameter name = value*.

```
$ exp
```

```
Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.
```

```
Username: system
Password:
```

```
Connected to: Oracle7 Server Release 7.1.3.2.0 - Production Release
PL/SQL Release 2.1.3.2.0 - Production
Enter array fetch buffer size: 4096 >
```

```
Export file: expdat.dmp >
```

```
(1)E(ntire database), (2)U(sers), or (3)T(ables): (2)U > 2
```

```
Export grants (yes/no): yes >
```

```
Export table data (yes/no): yes >
```

```
Compress extents (yes/no): yes >
```

```
About to export specified users...
User to be exported: (Return to quit) >
```

The Oracle Import utility is very similar to the Export utility. It is designed to read the files that are produced by Export and enable you to determine what data from these files is to be moved into your current Oracle instance.

Rather than go into the details of the parameters that you can pass in to these utilities, take a look at a few key ways you can use them to your advantage:

- ◆ You can export and import selected tables and users. This is a good way to back up only tables on which you may be performing major work.
- ◆ You do not have to be a DBA to work with Export and Import. You do need to have special privileges (DBA in Oracle 6 and special export and import privileges in Oracle 7) to export or import the entire database.
- ◆ One of the most popular features of Export is the Compress extents option. Let's say that you have a table that has run into a large number of extents. You wish to reduce it so that it fits in just one extent. If you export this table using the Compress extents option, drop it, and then import it with the data, you wind up with all the data in that table occupying a single extent. This procedure is covered in Chapter 27.
- ◆ There are a number of fancy features in Import that enable you to move data around or change the database objects.
- ◆ Import does not commit after every few records unless you set the `COMMIT` parameter to `Y`. It basically loads the entire file before issuing a commit statement. This can stress the capacity of your rollback segments if you import large tables. (I have waited over two hours before it gave me the message that it ran out of rollback space and then had to wait another two hours while it rolled back the transactions, so be careful.)
- ◆ Import (with some help from export) is an easy tool to use to move tables from one Oracle database instance to another.

This has been a quick overview of these two useful utilities. If you have further questions about Import or Export, the Oracle utilities user's guide provides chapters that explain command-line options and other features.

LOADING DATA FROM EXTERNAL SYSTEMS: SQL*LOADER

Some Oracle instances make all their own data and live in their own world. Others, however, have to get data from other Oracle and non-Oracle databases. The most commonly used utility to accomplish this function is SQL*Loader. SQL*Loader is designed to read text files that contain either delimited (with semicolons between all the columns of data) or fixed format (columns 1–20 contain last name, columns

21–35 contain first name, and so forth) It is not a pretty utility. You have to build control files, which are scripts that tell SQL*Loader how to read the input data, which tables and columns to assign the data that is read to, and how to insert the data. The good points about it are that you commit transactions routinely, thereby minimizing stress on rollback segments, and it works. I never underestimate the benefits of having a utility that works. The following code might be used in a control file:

```
load data
infile 'golfscor.dat'
insert into table golf_scores
(course          position(1:20) nullif =blanks,
total          position(21:23))
```

There are two basic options for getting data into SQL*Loader. The first, as shown in the example, is to store the data in a separate text file and tell SQL*Loader what that file is called. The other option is to store the data records at the end of the SQL*Loader control file. This sort of brings back the days of card decks where data cards were put at the end of programs, but it can be useful in certain circumstances.

A few other points about SQL*Loader to consider include the following:

- ♦ SQL*Loader can perform conversions as specified in the control file. This includes many of the basic SQL conversion functions (for example, text characters to dates).
- ♦ SQL*Loader can also perform some basic what-if logic to control the values transferred to the database. In the previous example, a null value is inserted for the course field rather than all blanks.
- ♦ SQL*Loader can produce a log file (whose name you specify in the command line) that records how the data has been divided up and the results of the loading operation. This documents whether any of the records were not loaded due to syntax problems, and so forth.
- ♦ SQL*Loader can optionally create files that store any data that was rejected due to having a bad format or because it did not match selection criteria that you input.
- ♦ SQL*Loader has a fast load option that enables you to bring in large quantities of data in shorter periods of time.

This section provides an introduction to SQL*Loader. It is one of the more difficult utilities to describe in a brief section because it has so many different options and parameters. The following is a sample of the command line that you use to activate SQL*Loader. In the Personal Oracle7 world, you get a fill in the boxes interface in which you input this data:

```
sqlload userid=system/manager control=golfscor.ctl log=golfscor.log
```

THIRD-PARTY PRODUCTS

There are a number of third-party products that perform some or all of the functions previously. The Oracle database market is a large one; therefore, software vendors have incentive to develop products that they believe are better than the Oracle products. Some stress convenient interfaces or additional automation capabilities. Try out any of these products for a while before you buy them. A good place to get a feel for the products that are available is local Oracle users' groups. Many of these groups invite vendors to demonstrate their products and provide briefings.

LOCALLY DEVELOPED TOOLS

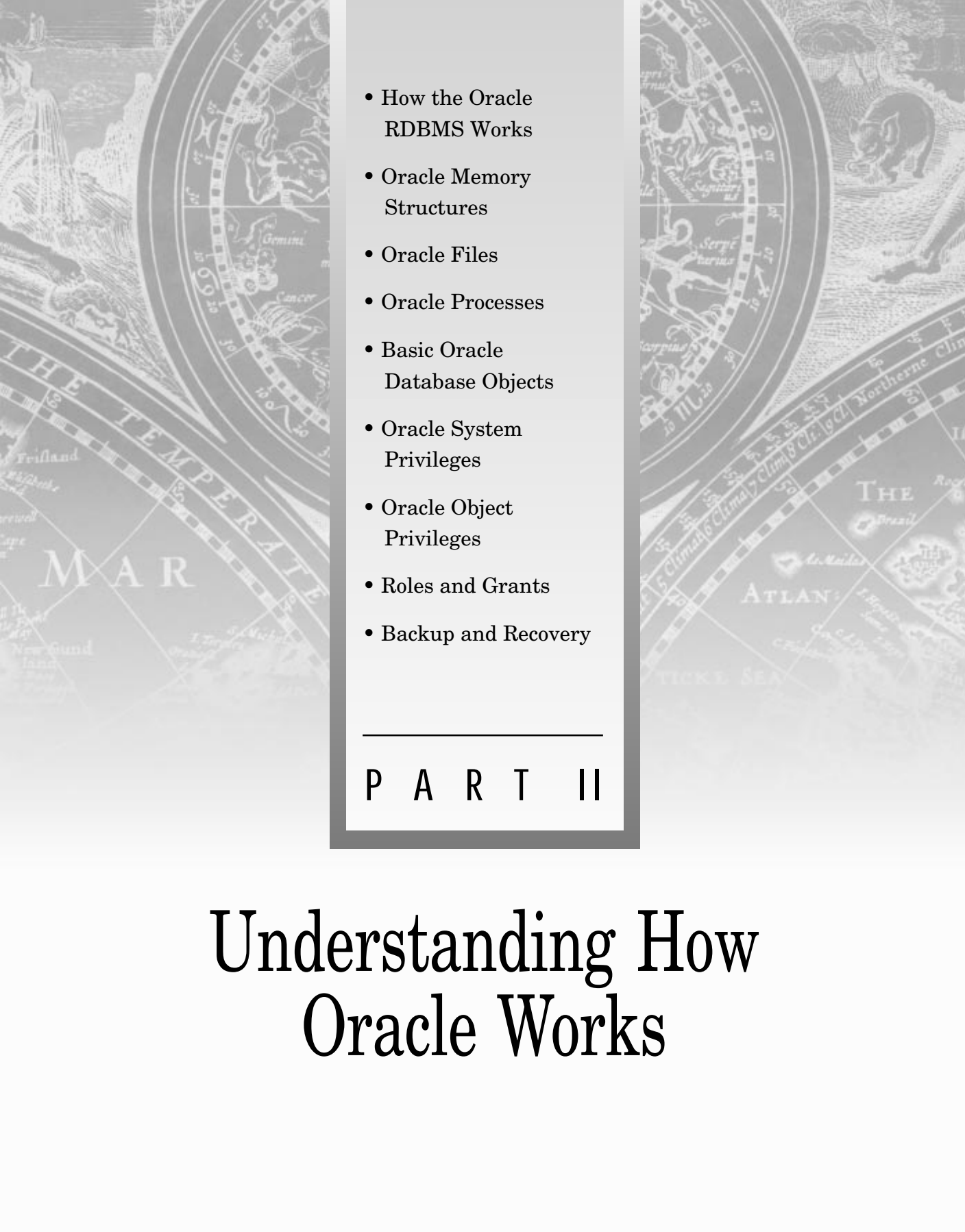
Almost everything that you need to monitor and maintain an Oracle database can be accomplished with SQL*DBA and SQL commands. Of course, the exact command syntax can be rather long and you may have to do a bit of research to find out what views and columns to access to find the information that you want. If you are a programmer at heart (or always wanted to try to be), you can build SQL and operating system scripts to store many of these commands for later use. The scripts contained on the enclosed disk are some of the ones that I commonly use to automate functions, such as setting up new users (it can be quite a long command if you want to specify default tablespaces and quotas) and running tuning reports on the database.

I always look at a particular task and try to figure out whether it is worth building into a script. This is usually a function of how often I perform the task weighed against how hard it is for me to type that command successfully at the command line. When I maintain large and active databases, I tend to develop more scripts. In my personal Oracle instance, I do not have any scripts other than those that I have brought over for testing as part of this book. This brings up a final point about scripts. There are a number of scripts available on Oracle forums on bulletin boards and from other DBAs. I normally hesitate about downloading software for fear of viruses, but if the source is fairly reputable (for example, the CompuServe Oracle forums) and they are SQL scripts (which are not as easy to pass a virus in as executable files), I will give them a lookover (scanning for suspicious statements) and use them if they look good.

SUMMARY

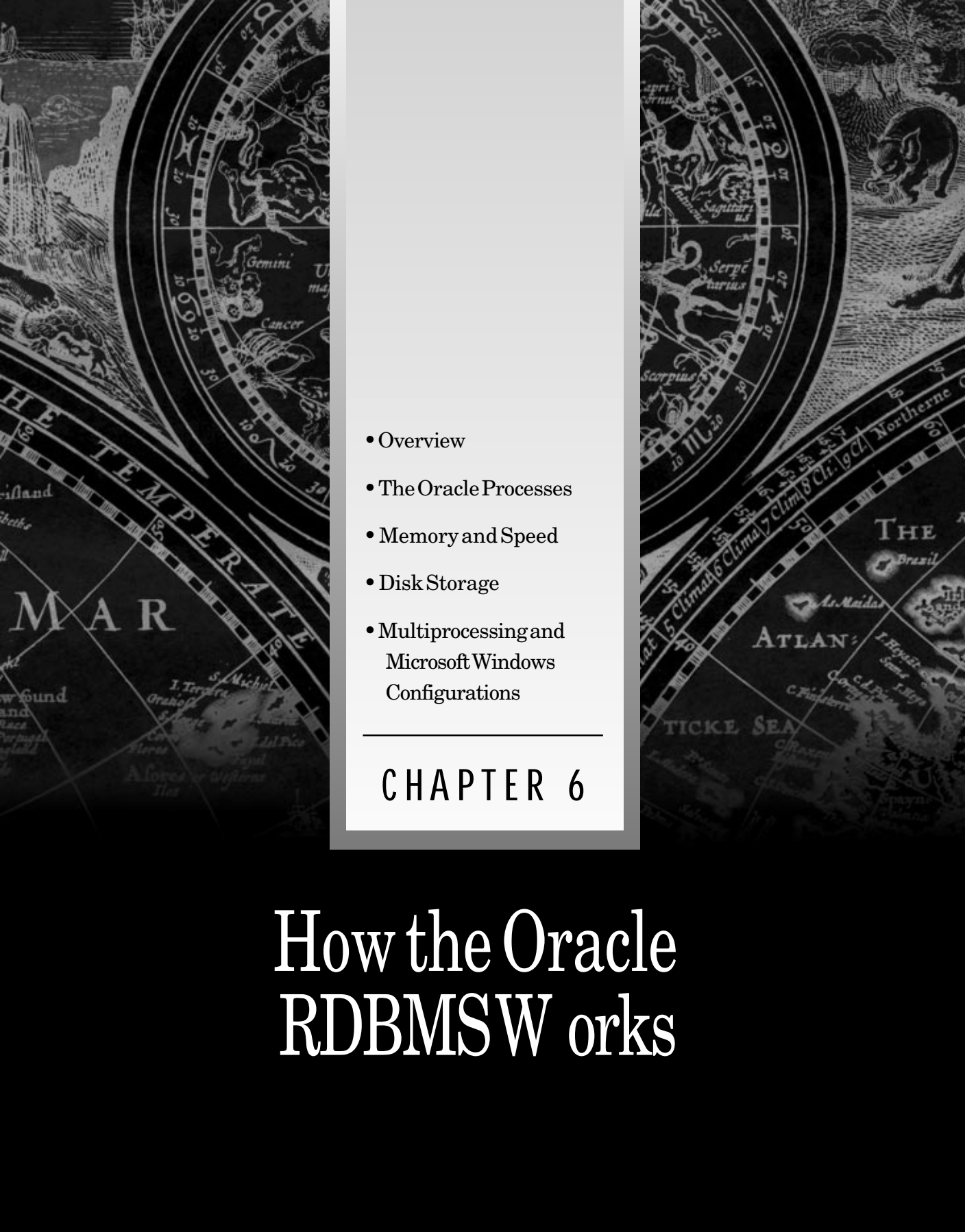
This chapter has presented a large volume of information for your consideration. There are entire books in the Oracle documentation set devoted to these utilities. Now you have a flavor of what these tools can do and an overview of how you use them. One final note: Because many of the functions that you perform using these

tools (such as exporting, dropping, then importing a table) can cause problems (for example, you mess up the export and then have nothing to import), try them out on a test instance if you have one. Although these utilities are not that difficult and most of you will get it right the first time, be sure before you try something on that critical production data.

- 
- How the Oracle RDBMS Works
 - Oracle Memory Structures
 - Oracle Files
 - Oracle Processes
 - Basic Oracle Database Objects
 - Oracle System Privileges
 - Oracle Object Privileges
 - Roles and Grants
 - Backup and Recovery

P A R T II

Understanding How Oracle Works

- 
- Overview
 - The Oracle Processes
 - Memory and Speed
 - Disk Storage
 - Multiprocessing and Microsoft Windows Configurations

CHAPTER 6

How the Oracle RDBMS Works

This book focuses on the job of the DBA and not the theory behind databases. The material in this section enables you to understand what happens when DBA tasks are performed. Think of this as getting comfortable with what is going on before you actually have to do anything. With the large number of platforms on which Oracle works and the never-ending series of upgrades and changes, there is no simple set of checklists that can be followed to be an Oracle DBA. Instead, you start with a basic understanding of what Oracle is doing, merge it with some tips based on experience from others, and combine it with your unique situation to determine how you are going to run your database. Most Oracle DBAs have longed for that simple checklist, but if it were that easy, anyone could do it and it would lower the salaries paid to Oracle DBAs.

This chapter provides a high-level overview of how the Oracle RDBMS works. Oracle is a rather complex beast, so this chapter focuses on understanding the pieces that make up this puzzle. The rest of the section delves into the individual components of the RDBMS. Remember, you do not have to understand enough of how Oracle works to be able to rewrite it. You need to focus on understanding the concepts of where data is flowing, where data is stored, and what processes are working on a given task. Then, if problems occur or the instruction manual is wrong, you will know what to do to correct the situation.

OVERVIEW

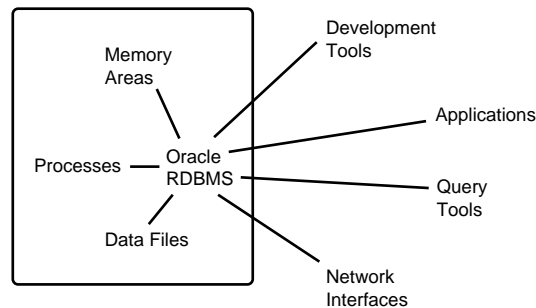
Oracle is primarily designed to work in multi-user, multi-tasking computer environments wherein the users are interacting directly with the operating system. There have been versions of Oracle that work in batch processing environments (for example, IBM mainframes) and single-user, single-task environments such as the PC. However, the basic architecture is designed to work well with operating systems such as UNIX, VMS, and so forth. There are a few key concepts about these operating system environments that you should know:

- ♦ These operating systems run multiple jobs or processes at the same time. The system cycles between the jobs and gives processing time to the processes that are ready to do something. Some operating systems enable you to control the priority of the jobs so that certain jobs get more time than others.
- ♦ The memory of these computers is divided into space reserved for the operating system, space reserved for the users, and space that is shared by multiple processes. The shared memory areas are not normally used by applications; however, as you will see, Oracle makes extensive use of shared memory areas.

- ◆ The disk drives of this system are shared between multiple users. Some form of security is usually present to control which users or groups of users can access data on particular disk drives.

Recall that Oracle is a complex set of products ranging from development tools to the database management system. This chapter focuses on the software products that Oracle has created to store and retrieve data—the relational database management system. A good way to start this discussion is with an illustration (see Figure 6.1).

Figure 6.1.
The pieces of the Oracle RDBMS.



One of the first concepts that needs to be understood about Oracle is the difference between an instance and a database. Think of this as your first Oracle-ism (those little quirks seen when you walk in the world of Oracle). A *database* is an organized collection of data stored in data files. An *instance* is the set of processes and memory areas that provide the capability of accessing this data. You need both to have a useful system.

The complexity and wonder of the Oracle RDBMS can be divided into three simple parts:

- ◆ Background processes
- ◆ Shared memory areas
- ◆ Disk data storage areas

The key is to understand how these areas work together before trying to understand how each one does its particular job. The easiest place to start is the disk data storage areas. Anyone who has worked with computers, from word processors to mainframe flat files to other database management systems, has stored data on some form of disk drive. Disk drives are relatively inexpensive places to store large amounts of data in a readily accessible place. When data is written to a disk area, it is not lost when you turn off your computer, which would happen if it were stored in the random access memory of most computers. Disks are the final resting place for your information.

In most PC database environments, such as dBase and Access, you interact directly with the data files using their DBMS commands. This works well for single-user, PC-based systems, but runs into problems for large, multi-user systems such as Oracle. For example, how would the system coordinate two users writing records to a single table and index? Oracle, like most other multi-user database management systems, uses a series of background processes to accomplish the data writing process. This also reduces the complexity of each of these processes, which is significant, considering how complex the system already is. These processes can be thought of as a series of specialists, each working on a particular task. For example, when a user writes changes to the database, these changes are placed in a memory area. When the memory area is ready to be written to disk, the database writer process transfers the changes from memory to the appropriate records on disk.

The exact number and configuration of the Oracle background processes depends on operating parameters, tuning parameters, and the optional packages selected by the database administrator. For example, if you want to keep all the files that log transactions, you start an archiver process that copies these log files to tape or another area on disk. If you want to have many input/output database write operations being performed at the same time, you start multiple asynchronous database writer processes (you set up the number as a configuration parameter). If you are starting to get confused about how many processes to start, don't worry. The basic configuration works well for most situations. Later chapters discuss when to start the additional processes to improve performance or provide additional services. For now, it is enough to understand the basic concepts of the background processes.

The final component in the Oracle database architecture is the memory areas. It is based on a very simple principle—memory is much faster to access than disk. When users need to write something, have them write it to memory and then go on about their business. When retrieving rows from the database, retrieve a few extra into memory and hope that the next row that the user needs is in that batch. A fair amount of the complexity of the Oracle RDBMS lies in the algorithms used to decide what to store in memory. Much of the tuning process involves working to achieve proper sizing of these memory areas. There's a section later in this chapter plus an entire chapter on the memory areas.

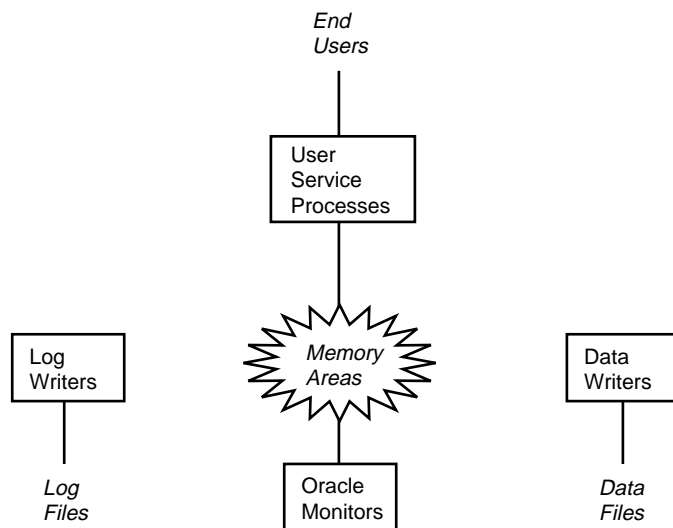
The user interacts with memory areas. These memory areas store data retrieved from and to be written to disk, which is the permanent storage medium. Background processes provide data retrieval and storage functions, linking the memory areas to the disks. There are background processes to perform other system monitoring and maintenance functions. However, if you understand the basic concepts in this paragraph, you're where you should be for now.

THE ORACLE PROCESSES

Now it is time to delve a little more deeply into the major components. The Oracle processes can be divided into four categories (see Figure 6.2):

- ◆ Processes servicing user requests
- ◆ Processes writing data to the data files
- ◆ Processes recording transactions in log files
- ◆ Processes monitoring the functioning of the database

Figure 6.2.
A simplified view of the
Oracle RDBMS
processes.



The first group of processes are those involved with servicing user requests. Think of them as your liaisons to Oracle. When you have a need for information, they go after it for you. When you have updates or inserts for the database, they store the transaction in the shared memory areas for later transfer to the data files. There are different types of user server processes, but for now it is enough to understand that they are there and that their purpose is serving as your interface to the Oracle database.

The next group of processes write data to the data files. Recall that when you write to the Oracle RDBMS, the data is stored first in a shared memory area. Oracle has one or more database writer processes that take the data from this shared memory area and write it to the data files efficiently. This creates free space in the memory area that can be used for other transactions and queries.

The third group of processes records transactions in log files. One of the major differences in a commercial-grade, multi-user database is that the capability of recovering from a failure such as the loss of a disk drive is very important. Oracle accomplishes this, in part, by recording each transaction (that is, inserting a record in a particular table) in a separate file from the data files. In the event of the loss of one of the data files, these log files can be applied to a copy of the data file retrieved from a backup to bring the data file up to the point where the failure occurred. More on this topic later, but it is important to understand the concept of these log files and that there are one or more processes devoted to taking transactions stored in the shared memory areas and moving them to these log files.

The final group of processes under our simple classification scheme is those that monitor the functioning of the database. You may already be worried about the complexity of the database as it has been described so far. Perhaps you looked at the next couple of chapters where the topics introduced in this section are discussed in more detail. Fortunately, you do not have to keep track of what is happening and manage the details of the database by yourself. Oracle has implemented a number of processes whose job it is to keep an eye on the database and correct problems when they occur.

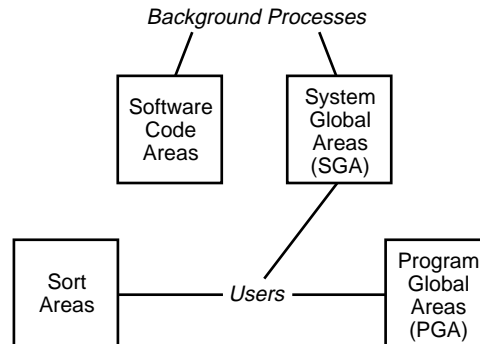
MEMORY AND SPEED

As mentioned earlier, shared memory is the key to speed in the Oracle RDBMS. It is also one of the most elusive areas to keep track of, and therefore it is important to understand. It is elusive because you can run operating system utilities to see which processes are running and you can look at the data files to see when they were updated, what their size is, and so forth. With memory, there is very little in the way of utilities that enable you to look at all of the shared memory components to see whether they are working well. The following are the various components or areas of memory used by Oracle (see Figure 6.3):

- ♦ Software code areas
- ♦ System Global Area (SGA)
- ♦ Program Global Area (PGA)
- ♦ Sort areas

The first memory area of interest is the one that stores software code. All that functionality from the Oracle database software needs a home. This is the location for it. On some operating systems, this software area can be shared between instances. Note that this is the software that runs the database itself, not the applications that you have written and are running.

Figure 6.3.
A simplified view of the
Oracle memory areas.



The System Global Area or SGA is the next memory area of interest. Actually, I like to think of this as the heart of an Oracle database. Your processes send transactions to this memory area and try to get data that has been cached here for speed. There are four key components stored in the SGA:

- ◆ The database buffer cache contains database records that are waiting to be written to disk or are available for reading.
- ◆ The redo log buffer stores a copy of transactions that have been applied to the database and are waiting to be written to the redo logs.
- ◆ Shared SQL areas store queries that have been parsed and are ready for execution. In many databases, users execute the same query many times, and performance is enhanced when these queries are waiting to be executed in memory. Some of this information is actually stored in the PGA (described in the next paragraph) in Oracle version 6.
- ◆ The data dictionary cache stores data about the database itself, its objects (tables, views, and so forth), and users.

In contrast to the SGA, which contains information shared by everyone in the instance, the Program Global Area or PGA stores information that is used by a single-user process. When someone creates an Oracle session (Oracle's way of saying that the user connects to a database instance), Oracle creates the PGA. The PGA contains stack space, and in many cases, information about the user session. You learn the differences in PGA contents a few chapters from now.

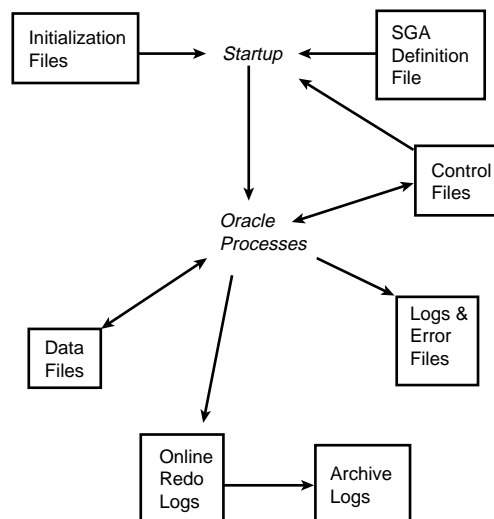
Finally, sort areas can greatly improve the performance of one of the most common database operations—sorting. Very few people want to review data records in the order that they were input (or in Oracle's case, the order that Oracle chooses to store the data). Most applications have some form of ordering to them (you sort a list of names alphabetically, sort dates, and so forth). Using the logic that memory access is much faster than disk access, Oracle likes to store a list of values to be sorted in memory and perform the sort there. That is the purpose of the sort area.

DISK STORAGE

After that glowing discussion about the wonders of memory, it is time to address the somewhat slower, but inexpensive and nonvolatile storage mechanism found in most Oracle instances—the magnetic (hard) disk drive. These little jewels enable us to build those multi-gigabyte (or in some cases kilobyte) store houses of corporate wisdom. This section provides a brief introduction to the types of files that Oracle employs in its battles with information overload. An entire chapter is devoted to the subject later in this part of the book, but for now, you need to look at the seven types of data files used by the Oracle RDBMS (see Figure 6.4):

- ◆ Data files
- ◆ Online redo log files
- ◆ Archive log files
- ◆ Initialization files
- ◆ Control files
- ◆ SGA definition files
- ◆ Oracle processing log and error files

Figure 6.4.
The basic types of
Oracle data files.



The easiest place to start is the data files. These contain exactly what you would expect to find in a relational database management system—a series of records of data arranged in tables. There are a number of other types of database objects

(indexes, views, and so forth) that you will find in these files, but for now, it is enough to understand that this is where you will find your data. One interesting point to note is that Oracle now enables you to store software within these database files (in the form of packages and procedures).

The next set of files is the online redo log files. The concept of these files is simple—record every transaction made to the database in a file separate from the main data files. These can be used to recover all changes made to the database in the event that a data file is damaged. Oracle uses several of these files so that when it gets done writing to the last file in the series, it begins overwriting to the first online redo log file. An obvious question is, how can you recover from a damaged data file if you overwrite one of your online redo log files?

The answer lies in the archive log files. Their function is simple. When you finish writing to an online redo log file, a background process makes a copy of that redo log file to a separate file that is given a unique sequential number. This file can be placed on a magnetic disk drive or a magnetic tape. Then, if you want to recover a data file that has been destroyed, you get a copy from the backup tapes that you have been religiously making and apply all transactions in the redo and archive log files that have occurred since that backup was made.

Now that data storage has been taken care of, it is time to investigate some of the supporting files that Oracle uses. The initialization files are the equivalent of Microsoft DOS's `autoexec.bat` and `config.sys`. There are two files in this set, although you could get by with one or even more than two. These files are called `init.ora` and `config.ora`. The exact name of a file for a particular instance has the ID of your particular Oracle instance in it. For example, if your Oracle instance has an ID of `test`, your files are called `inittest.ora` and `configtest.ora`. With that out of the way, the purpose of these control files is to specify the following startup parameters to Oracle:

- ◆ Values for all those tunable parameters that Oracle uses to improve performance
- ◆ Locations of the control files and archive log files
- ◆ Locations for some of the log and error files

The next set of files that helps Oracle keep track of what is going on is the control files set. These files keep track of where the data files are and also record the number of the latest transaction applied to the database. This latter feature helps Oracle during recovery to know how many transactions need to be applied in order to come up to date. These files are stored in a binary format that you cannot read using standard operating system text file commands (such as `more` on UNIX or `type` on VMS).

Another binary supporting file is the SGA definition file. You will have one of these for each of the Oracle instances that you create. This file tells Oracle some details about creating the SGA on startup. Most of the DBAs that I have worked with are either not aware or are only vaguely aware of these files. This is a good sign for you new DBAs, because it means that you do not need to worry about these files.

The final type of Oracle file that you will deal with is the log and error files. Oracle is designed to be a production, business-grade database. Therefore, when problems occur, you need to determine what is wrong quickly and get it fixed. To support this need, Oracle records all major database activities into a series of log files. In addition, when Oracle's monitoring processes detect a problem, they write as much relevant data about the problem as they can find to a series of error logging files (called *trace files* in Oracle). In this way, the DBA can look at a few files and determine what happened.

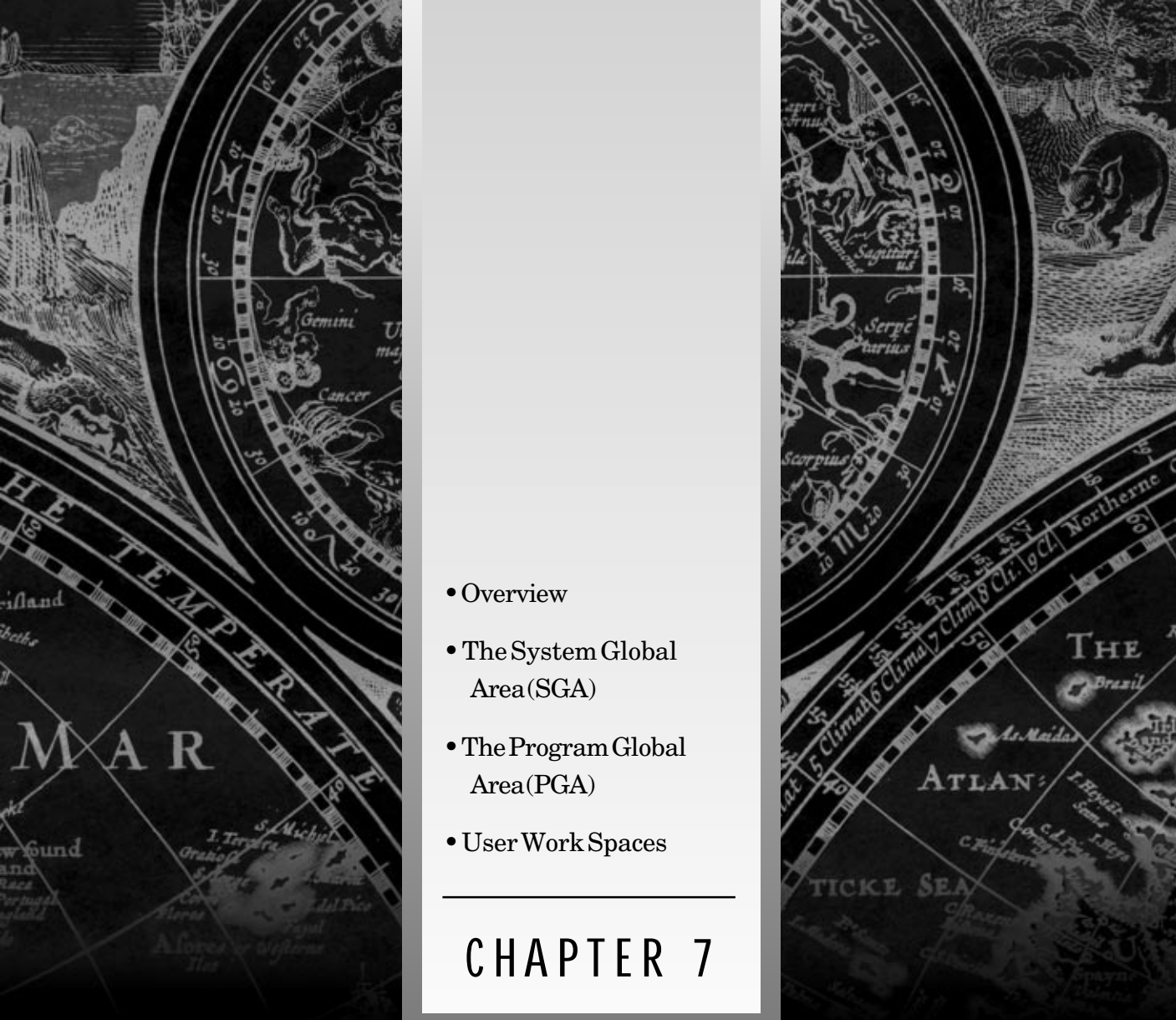
An interesting sidelight while on the subject of Oracle files is the task of finding all these files on your system. Many of the DBAs who I have come across do not know where all these files reside. This makes troubleshooting much more difficult. Therefore, in the chapter on files that is coming up soon, you will learn how to find all these files on your systems.

MULTIPROCESSING AND MICROSOFT WINDOWS CONFIGURATIONS

The preceding discussion covered the majority of Oracle installations in the field today—those on multi-user operating systems. Those of you who are working with the old Oracle for DOS products or the new Oracle for Microsoft Windows 3.1 know that these system cannot have a number of background processes running continuously in the same manner that UNIX or VMS do, for example. Those who use Oracle on such systems as IBM's MVS know that the terminology and implementation have to be a little different on these radically different systems. However, this book presents this discussion in Oracle's native environment (UNIX and VMS) because although the names of the disk drives (which IBM calls DASD for direct access storage devices) and process configuration may change, the concepts remain the same. Even though you have a single program icon under MS Windows that is minimized on your desktop for the Oracle RDBMS, the logic of having something writing logs when needed and something writing database records from the SGA remains that same. The operating system's unique concepts are covered in the chapters that follow. For now, work on understanding the general concepts rather than worrying about the details.

SUMMARY

This chapter was designed to give you the overview—from 30,000 feet—of the Oracle relational database management system. You should now be comfortable with the general concepts of files, memory, and processes. The chapters that are coming up dive into these topics in more detail. Remember that you do not have to absorb it all at once. Just understanding the overview that was presented in this chapter puts you ahead of many DBAs, and they perform their day-to-day tasks well. Of course, your goal is to be more than just okay when it comes to database administration.

- 
- Overview
 - The System Global Area (SGA)
 - The Program Global Area (PGA)
 - User Work Spaces
-
- ## CHAPTER 7

CHAPTER 7

OracleMemory Structures

Having completed the introduction to the structure of Oracle that was presented in the last chapter, it is time to dive deeper into the mysteries of Oracle. This chapter presents the level of detail needed by database administrators on Oracle memory areas. An entire book could be written on the subject, but the DBA does not need to know those details. This chapter focuses on the topics that have practical uses, such as the following:

- ♦ In what memory areas does Oracle store various types of information?
- ♦ What are the impacts of inadequate space for the various memory areas?
- ♦ How are the values of the storage parameters for the various memory areas set?

OVERVIEW

A good starting place is the overview picture of the Oracle memory areas that was presented in the last chapter. Refer to Figure 6.3 in Chapter 6.

Before getting into the details of what each of the memory areas contains, it is useful to step back and consider what you need to put in memory to make a database management system work efficiently. This depends somewhat on the designer's preferences and style, but generally speaking, a multi-user DBMS that wants to use memory to achieve speed would want to put the following into memory:

- ♦ Rows of tables that have been added or changed. These are written to memory first so that the user process can go on without waiting for the disk to receive the data.
- ♦ Rows that are "close" to the row with which the user is currently working. In most applications, it is likely that the next row that the user will want will be close to the current row. Because most operating systems read entire blocks (512 bytes or more) from disk drives, if you save these extra retrieved rows in memory, you can get lucky and have the next row needed waiting in higher-speed memory.
- ♦ Space for the application programs that are running. Computers execute instructions stored in memory. If you want your general ledger application to work, you need to have memory space available for it. This same concept applies to storing code for the various Oracle background processes.
- ♦ Administrative space to keep track of who is logged in, what needs to be done, and so forth.

- ◆ Information being passed between Oracle and user processes. These messages enable the many processes involved with Oracle to coordinate their activities.

One of the considerations that you will run across is the use of real and virtual memory. For those who are not used to these terms, real memory consists of the random access memory (RAM) chips of the computer. This is the memory that gives all the speed benefits that we have been discussing in this chapter. Most multi-user computer systems, and even Microsoft Windows and the Macintosh operating system, provide what is known as virtual memory. Virtual memory is a section of a disk drive that is allocated to storing the overflow of the real memory area. This enables you to have more processes running than would be possible if you were limited to storing these processes in real memory. It works especially well when you have a number of processes that run infrequently, and therefore there is not much of a performance impact when they need to be swapped out to disk. The swapping out process transfers the program and data as they reside in memory to a special area on one or more of the disk drives, thereby freeing up real memory to be used by other processes. In Oracle's case however, you really want all of Oracle's memory areas to be stored in real memory, because speed is the whole purpose of this architecture.

Oracle divides up its memory areas into several sections. The System Global Area (SGA) holds all the common database storage areas (transactions being buffered, data dictionary information, and so forth). The Program Global Area (PGA) stores data related to your individual process. Spaces related to the software being run by the Oracle background processes and user processes are included. Finally, there are areas related to a user process that are used for sorting data. This is a nice, clean division of function and if you can understand these basic concepts, you are well on the way to understanding enough of how Oracle works to be an effective DBA.

An important concept to understand as a DBA is the control of how memory regions are set up for sharing. Most memory areas (such as the one allocated to your shell when you log on to UNIX or the one for Microsoft Windows) are not sharable. You have sole access to that memory and that is the way it should be. Shared memory is the exception to the rule and it usually takes some special permissions from the system administrator to enable this memory area to become sharable. The exact method for setting this up varies from operating system to operating system. The RDBMS software provided by Oracle is normally set up so that when it creates a new database instance, it has the permissions necessary to set up the memory areas as sharable without your having to take any overt actions.

Note

Something may have happened if you get a message indicating the SGA is not available, but the following are true:

- ◆ You see the Oracle instance running properly.
- ◆ Only the user who started the Oracle instance (oracle) can access Oracle successfully.

Check with the system administrator regarding shared memory permissions.

You may be curious as to how you can control all these wonderful memory areas to set them up properly for your particular application environment. The answer lies in the combination of initialization files (discussed in detail in Chapter 8) and the default parameters that Oracle has established for its products on your operating system. Let's start with the default settings. Oracle takes into consideration the average configuration of a particular type of computer and comes up with a set of default values for those parameters that determine the size of the SGA, PGA, and sort areas. Obviously, the configuration for the Microsoft Windows version of Oracle is set up to be much smaller than that of a large VAX computer. The settings of these parameters can be overridden by values entered in the initialization files. So, if your application does a lot of large sorts, you can increase the size of the sort area over that of the default. (Chapter 8 presents how to make entries in the initialization file to change the values of these parameters. Chapter 32 shows you how to analyze your instances to determine how to adjust (or tune) these parameters.)

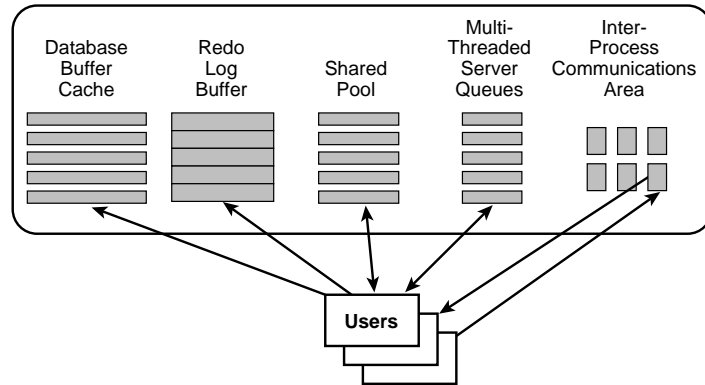
THE SYSTEM GLOBAL AREA (SGA)

The SGA can be thought of as the heart of Oracle itself (see Figure 7.1). It holds changes that you make to your database until a process is ready to write the data to disk. It stores things that can help speed up your access to data. Without the SGA, you do not have an Oracle database. Anything that is this important deserves some attention from the DBA. In this section, you explore an overview of what the SGA does. Although the SGA usually works just as well whether you understand it or not, this knowledge can come in handy some day when a performance problem arises and the cause is not obvious.

As with most of the topics in this book, it is easiest to break the SGA into its components and discuss each of them individually. The only time that you have to be concerned with how the pieces fit together is when you calculate the total amount of memory space required for the SGA. Sizing is discussed later in this section. For now, the pieces of the SGA are the following:

- ◆ Database buffer cache
- ◆ Redo log buffer
- ◆ Shared pool
- ◆ Interprocess communications area
- ◆ Queues for the multi-threaded server (when used)

Figure 7.1.
The System Global Area (SGA).



The database buffer cache stores records from various tables within the Oracle database (actually, because reads and writes are in blocks, it stores the entire database block that contains the records that you are working with). These blocks contain either rows that have been read from the data files or records that need to be written to the data files. In effect, the database buffer cache is a way-station for data that sits between the users and the data files. As with most hotels, there is a limited capacity for accommodating these guests. Therefore, it is important for the DBA to understand how Oracle determines how long an individual record block gets to stay.

The easiest decision on how long a block of records needs to stay in cache involves new or modified records that are waiting to be written to disk. They stay in the database buffer cache until they get written by the database writer process. It would not work if Oracle got to throw out data blocks when it felt too full. For the remaining buffers, which have been read in from disk and not modified, Oracle has to decide which ones to keep and which ones to get rid of. Many systems use a “first-in, first-out” method. This may seem fair, but it is not efficient for most database circumstances. Many records are accessed very frequently (a common lookup table, for example), so it makes more sense to keep those that are used in memory and drop those that are not used. The algorithm that implements this concept is called least recently used (LRU for those of you who are fond of acronyms).

When a record is called into the database by an Oracle query, it is stored in the database buffer cache and is placed at the most recently used end of the list of records. Every time a record in the database buffer cache is used, it gets promoted to the most recently used end of the list. When a query needs space to store data that has to be pulled off the disk drives and all of the buffers in the database cache are full, it overwrites buffers that are at the least recently used end (top) of the list. If records that are overwritten are needed again, they are called in from disk.

This process works well and saves you time by storing data in fast memory to avoid slow data transfers from disk. Let's challenge you with an opportunity to save even more time and show your database that you know more than it does. Unless you have an extremely large amount of real memory, you have to limit your database buffer cache to some reasonable value. The LRU algorithm therefore looks at data from a short-term (maybe several minutes) point of view. You, however, know your applications and users very well and understand the "big picture." You know that there are several relatively small tables that are used very frequently (perhaps they are used to look up values to make calls against your big data tables). The LRU algorithm may call these rows into the database cache at the beginning of one of these queries, but they are eventually pushed out as you move in row after row of data from that large main table. When the next user issues a query, you have to start this process over again. However, Oracle 7.1 and later gives you the option of caching a table in memory. This has the effect of bringing the entire table into memory and keeping it near the most recently used end of the database buffer cache. You have to be careful because too many cached tables can fill up the buffer cache making that slow disk access a necessity for every row read. However, used with good judgment, this can be a useful tool to speed up databases that are suited to benefit from it.

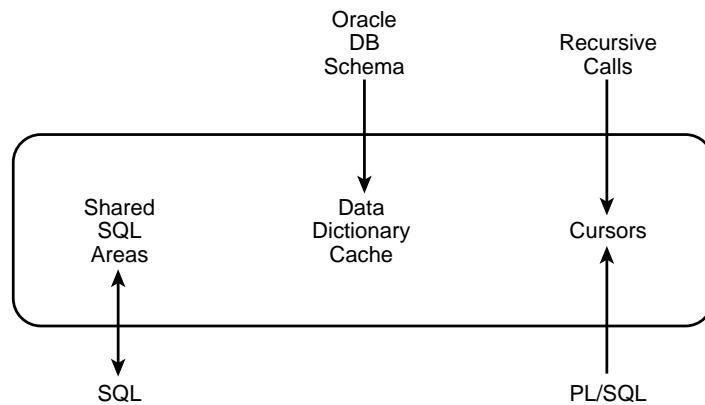
The next area of the SGA to explore is the redo log buffer. As discussed in Chapter 6, updates to the Oracle data files are recorded in the data files and also separately in the redo log files. This duplicate record enables recovery in the event of the complete loss of data files. The logic goes that if you do not want to wait for the data records to be written one at a time to disk, you do not want to hold up the transaction for the redo log entry to be written to disk. In fact, because most operating systems read and write blocks that are several kilobytes in size, it is more efficient to queue up an entire block of data and then write it. That is the purpose of the redo log buffer.

Because the redo log buffer is designed for writing as opposed to reading, it uses the first-in, first-out approach to storage. Redo log entries are added in the order that they are received and the log writer process of Oracle comes along and takes one or more blocks of these records and writes them to the online redo log files. The redo log buffer then serves as the way-station for redo log entries, sitting between the Oracle process that creates the database update transaction and the redo log files on disk.

The third of the five areas within the SGA is the shared pool. You have seen the benefits of storing data that has recently been read from disk or needs to be written to disk (data and log files) in memory. The developers of Oracle looked at the services performed by a database and came up with a number of other things that should be stored in memory areas to improve overall performance. The shared pool contains three of these performance improving memory areas (see Figure 7.2):

- ◆ Shared SQL areas
- ◆ The data dictionary cache
- ◆ Cursors

Figure 7.2.
The shared pool
memory areas.



The first of these shared pool areas stores what Oracle calls shared SQL. The structured query language (SQL) is a standard way of interacting with the database that is common to Oracle, IBM's DB2, Informix, and a high percentage of the multi-user databases that are on the market today. An example of an SQL statement would be the following, which pulls back all the rows and columns from the fictitious `big_payroll` table.

```
select * from big_payroll
```

Oracle performs a fair bit of work to service this request. For example, Oracle needs to figure out what are the columns in the `big_payroll` table, whether there is an index that can be used to speed the query, whether the user has access privileges to this data, and many other bits of information to get the data for the user. In many databases, there are a series of queries that are frequently used (an order entry system may routinely pull up a list of valid product codes, for example). The shared SQL area stores the analyzed (or parsed) query for any user to access. Queries are stored in this area, which is also referred to as the library cache, using a modified form of the LRU algorithm.

The second part of the shared pool area is the data dictionary cache. In all queries and transactions with the database, the database management system needs to determine where the data is. This includes such details as what the object (table) names are, what tablespaces they are located in, where their records are within the data file, and what their columns are. In addition, database security requires Oracle to verify that the user has the appropriate access permissions for that database object. All this dictionary data information is stored within various system tables in the database. Such frequently used information is a wonderful candidate for caching. The data dictionary cache stores this for rapid access by the RDBMS.

Sometimes the data dictionary cache is referred to as the row cache. To me, row cache would be what I would call the area that stores rows of data from the database (the database buffer cache). However, I am not the designer of the system or the names used to refer to the components of the system.

The final area within the shared pool contains the cursors. Cursors are actually stored within the shared SQL area, but they are conceptually different so let's look at them separately. The previous `select` statement displays the results of its query on the screen. What if you want to store the results in a memory area so that you can manipulate the data in some way? Basically speaking, that is what a cursor does. It stores data retrieved from the database for further processing. Oracle creates its own internal cursors (known as recursive cursors) when it performs statements such as `CREATE TABLE`. Statements such as this cause a number of updates to various data dictionary tables (which are a series of SQL statements, referred to as recursive calls, that Oracle takes care of behind the scenes). Both of these forms of cursors take up space in the shared SQL area and should be considered when you are sizing the shared pool. You should also note that part of the storage required to support cursors is in the PGA (discussed in the next section). The PGA contains pointers to the SGA and perhaps the entire parsed SQL statement, depending on your version of the Oracle database and how you are accessing it.

With the shared pool covered, it is time to move on to the fourth area within the SGA. This area does not have a fancy name. It is used to store messages transmitted between various processes associated with the operation of the Oracle database. Many features, such as locks applied to objects by a process, are conveyed using this area. The good news is that there are no special tuning parameters and intervention required by the DBA for these message areas. As long as there is sufficient size in the shared pool, you're set.

The final component that can take up space within the SGA is space for the multi-threaded server queues. One of the difficulties in writing a book that discusses a tightly integrated system such as Oracle is that sometimes you have to introduce a

topic before its time. The multi-threaded server is explained in detail in Chapter 9. Basically, the multi-threaded server is a feature of Oracle 7 that enables users to share the memory areas and processes used to connect user applications to the Oracle database. The logic is, why waste memory duplicating these relatively common areas for perhaps hundreds of users? For purposes of this chapter, all you need to know is that if you are using the multi-threaded server option, these memory areas are stored in the SGA and affect its size.

As DBA, you may be the only one who is aware that there is such a thing as the SGA; therefore, you will be the one to manage it. Many business book writers argue that you cannot manage anything that you cannot monitor, so the first task is to monitor the SGA. My favorite way is to issue the following command in SQL*DBA:

```
show sga
```

This command produces a result similar to what is shown in the following example. Those of you familiar with large Oracle databases on such machines as Sequent, Sun, and VAX may wonder why the numbers are so small. This printout comes from my Personal ORACLE7 Release 7.1.4 system. Do not worry, most of my work experience is with large (up to 135G per instance) Oracle systems on UNIX and VAX servers. I use screen prints from my PC-based system because that is where my word processor is located. This command works the same on the large boxes (although most of them have much larger memory areas than I could afford for my PC):

```
SQLDBA> show sga
Total System Global Area      4767460 bytes
      Fixed Size                36432 bytes
      Variable Size           3846292 bytes
Database Buffers              819200 bytes
Redo Buffers                   65536 bytes
```

The final issue to discuss regarding the SGA is how to control it. In Chapter 32, you learn how to tune Oracle instances. For now, know that there are variables within Oracle that you can set using initialization files. These control the space allocated to the redo log buffer, database buffer, and other areas within the SGA that are discussed in this chapter. You will not be ready to alter these parameters until you learn how to determine *when* you need to adjust them in Chapter 32.

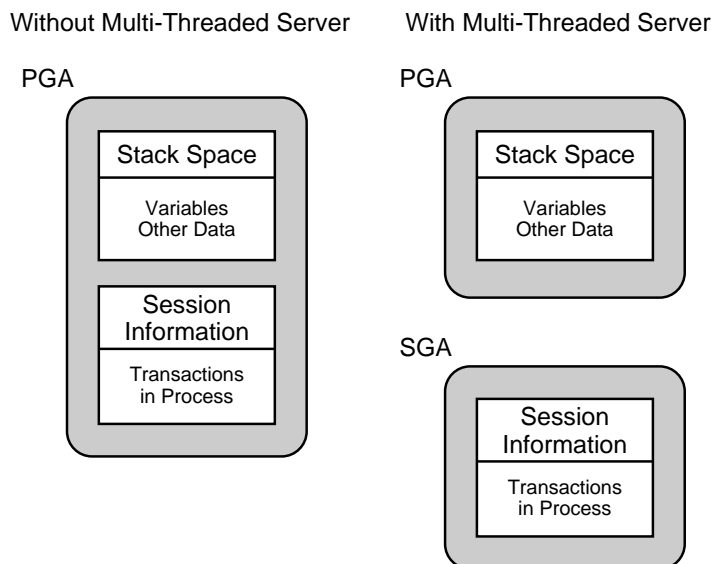
THE PROGRAM GLOBAL AREA (PGA)

The System Global Area provides memory for those things that all users need to share. There are several things that users need to keep to themselves and that is the purpose of the Program Global Area. All users are allocated PGAs when they connect to the Oracle database. The size of this memory area is fixed by the operating system and remains constant as long as the users are connected to Oracle.

For Personal Oracle users and those that have a small number of users, this is a workable arrangement. However, many of the larger transaction processing systems may have dozens or hundreds of users connected to the database at a given time. When you map out the large amounts of memory required for the SGA on such large databases and then add in space for each of the user processes and the operating system itself, there is not much space available for a large number of PGAs.

That is where the multi-threaded server comes into play. In most situations, there may be a large number of users logged on the database computer at a given time, but a much smaller number is actively executing queries or updates. Most of the users are either reading the outputs of the system, typing at rates that seem slow to the computer, or just thinking. Oracle's multi-threaded server is designed to allocate a specific number of spaces for the information that is involved with a particular transaction. This information includes the private SQL areas and other such items that relate to a particular question or update. The users still retain space, known as stack space, dedicated to their individual sessions to hold variables and other data associated with their work. The data stored in the PGA with and without the multi-threaded server is shown in Figure 7.3.

Figure 7.3.
The contents of the
Program Global Area
(PGA).



A few final notes about the PGA seem appropriate at this point:

- ♦ The PGA is owned by a single-user process and only that user can read from or write to it.

- ◆ If you do not use the multi-threaded server and there is not enough memory available on your computer, you will receive an error message from Oracle to that effect.
- ◆ If you are running a client-server configuration, the PGA for a user will be allocated on the machine that acts as the database server.
- ◆ Some literature refers to the PGA as the Process Global Area rather than the Program Global Area. Either way, it works the same.
- ◆ The size of the PGA varies between versions of Oracle that run on different operating systems. Three parameters affect the size of the PGA in a given instance: the number of open database links, the number of database files, and the number of log files.

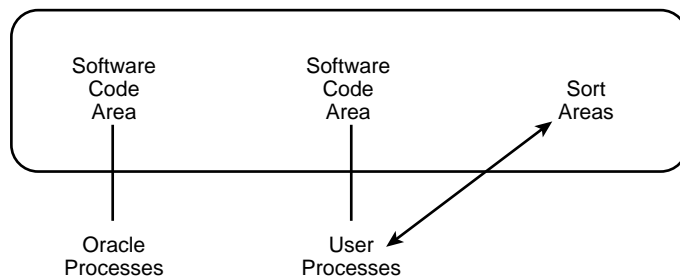
This is enough for most DBAs to understand about the PGA. DBAs typically do not have to do much with the PGA, unless they are running short on memory in their computer system. Because the parameters mentioned are controlled by application needs, the DBA usually has to acquire additional memory or use the multi-threaded server option to solve problems with PGA memory limitations.

USER WORK SPACES

So far you have explored the memory areas that Oracle uses internally to perform database services for the user. Where are the actual applications that are designed to perform specific processing tasks? Every application needs memory areas in which to run—work spaces. There are many important bits of data stored in this area (see Figure 7.4):

- ◆ Software code areas for Oracle processes
- ◆ Software code areas for user processes
- ◆ Sort areas

Figure 7.4.
The user work spaces.



The memory space used to store the Oracle application software code is owned by the Oracle user and should be protected by the operating system from modification

by other users. The areas are usually installed in a shareable fashion so that users can access parts of the software when needed. It is usually desirable to have one copy of the Oracle software available to multiple Oracle instances on the same computer. Certain operating systems such as MS-DOS do not have the capability of sharing application code. The size of the Oracle application software code area usually changes only when the software is upgraded.

The next software area to discuss is the memory space used by regular user processes. Most of the Oracle tools and utilities that the DBA will work with are shareable. However, most custom applications require separate work areas for each user. Each of these user areas is relatively small when compared with the SGA, for example; however, the DBA and system administrator may need to keep an eye on these processes when there is a large number of users on the system. Unlike the SGA, it is not an extreme performance problem if inactive user processes are swapped out to disk.

The final user memory area to consider is the sort area. One of the most common clauses in a SQL select statement is the order by clause. Very few people want to have to search a long list of output in the order that Oracle chooses to store it physically. Therefore, most applications perform a fair amount of sorting. The speed of memory is especially noticeable when performing sorts. It can easily take five to ten times longer (or more) for a sorted query if it will not fit in memory. However, especially in large data warehouse Oracle instances, you may have to sort data that will not fit within the physical memory of the system; therefore, you have to live with disk-based sorts.

The sort area size varies depending on application needs. Its maximum size is controlled by the `SORT_AREA_SIZE` parameter set either in the initialization file or the Oracle default for your operating system. Once the sort is completed, the size of the sort area is reduced to the value specified by the `SORT_AREA_RETAINED_SIZE` parameter. If the entire set of data to be sorted does not fit within the sort area, the data is sorted in chunks that do fit into memory and then the chunks are sorted together.

SUMMARY

You do not need to delve too deeply into the arcane technical internals of the Oracle database to be a good DBA. You should understand enough about the Oracle memory areas to be able to answer questions such as the following:

- ♦ When a tuning report indicates that the redo log buffer needs to be increased, what effect does that bring about within the system (SGA size)?
- ♦ When you get error messages indicating that the system could not allocate space for the PGA, what are your alternatives?

This chapter showed what the memory areas are that Oracle needs to run efficiently and the contents of these memory areas. From this discussion came the impacts (usually speed, but occasionally error messages) if these areas are not large enough. Finally, you explored how the size of these areas are altered, although it will not be until Chapter 32 that you learn a complete approach to database tuning.

The next chapter discusses the second of the three major parts of a working Oracle database instance—the disk files. Memory is the way-station for most information processed by the Oracle instance, and the data files are the final resting place for this data. There is a wide array of files used and most DBAs do not know about all the files that are out there that can affect them. Chapter 9 covers the third part of the Oracle instance—the operating system processes. After you have learned about these fundamentals, the following chapters will show you the objects, privileges, and backup options that DBAs work with on a routine basis.

- 
- File Locations
 - Data Files
 - Redo Log Files
 - Archive Log Files
 - Control Files
 - Initialization Files
 - Log and Trace Files
-

CHAPTER 8

Oracle Files



For a typical PC software product installation, you place a few dozen files in one directory or a series of subdirectories under one main directory. You know where everything that you need is and the world is wonderful. This chapter deals with the files of the Oracle database world. Here there are hundreds, perhaps thousands, of files that are scattered over a number of directories. This chapter shows the overall picture of where the files are stored and then highlights those files that the DBA usually uses.

It is not as bad as it may seem. There are an enormous number of files out there in the Oracle system. Each of the Oracle add-on packages (Oracle Forms, SQL*Net, and so forth) add many more files to that collection. However, from a DBA point of view, you can divide the many files into the following categories:

- ♦ *Oracle software.* These are the files that are purchased and installed to make the system work. These files make up the vast majority of the hundreds or thousands of Oracle files on your system. The good news is that, typically, you need to concern yourself with these files only at installation time. If they do not load and link correctly, your only recourse is to call Oracle support and have them help you get things working (unless you feel like rewriting major portions of Oracle in your spare time).
- ♦ *Data files.* When you first create an instance, you have only a few data files. In fact, you can have a single data file under earlier versions of Oracle (6 and below). In Oracle 7, you get at least four data files by default. If you have very large databases that span many disk drives, you may have dozens of data files. These are a relatively easy resource to manage because there are database views to enable you to locate the files quickly. You typically perform size maintenance on your data files more often than on any other type of Oracle file.
- ♦ *Redo log files.* You have only a handful (normally about four) of redo log files on your system. There are views to help you locate these files from within the database. These files typically need maintenance only when you are changing their size or moving them for enhancing performance.
- ♦ *Archive log files.* These will be your biggest concern when operating a database in archive log mode (which is explained in more detail later). If these files fill up their destination device, it stops all updates and inserts to the database from happening and perhaps crashes the instance. It is definitely enlightened self interest to know where your archive logs are and to monitor their size and clean them up routinely.
- ♦ *Control files.* If you are lucky, you may never bother with your control files after you create an Oracle instance. These files are used to help Oracle start up by providing configuration information and status. You alter their contents when you modify the database configuration, but Oracle takes care of this behind the scenes.

- ◆ *Initialization files.* These files store the information that is needed to start the system. They contain the locations for the control files and all the tuning parameters that you set to adjust the performance of your instance. You will typically bother with these files only when you get into tuning the Oracle database.
- ◆ *Log and trace files.* Many books do not emphasize these files. I was a pretty experienced DBA before I found the log file used for SQL*Net (which had grown to 1.4M because no one knew that it was there to clean out). However, these little jewels are your keys to troubleshooting major problems such as a crash of the instance. The major Oracle processes are designed to leave a last message on the system when they encounter problems and are crashing. The system actually has processes that monitor the activity of other processes and record messages when they detect that something is wrong.
- ◆ *Audit files.* Oracle enables you to audit certain events that occur in the database (see Chapter 31). You can store the record of activities either in the database or in a file that you specify in your initialization file.

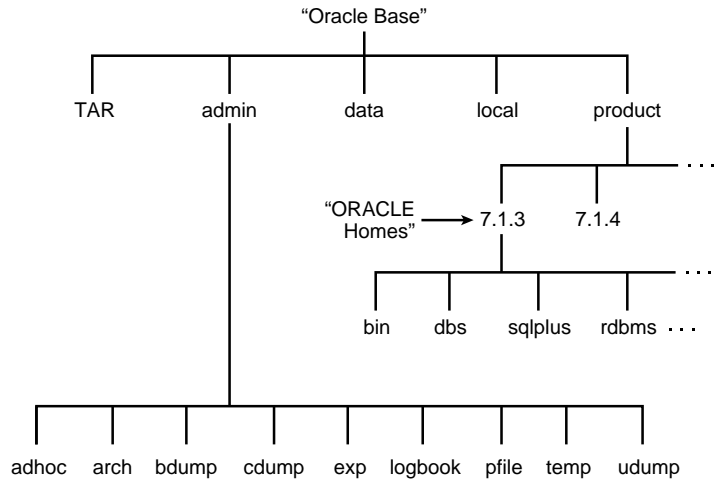
From a DBA perspective, we have just narrowed that list of hundreds or thousands of files associated with Oracle down to a list of a few dozen. When you consider that once you understand one control file or log file, you understand them all, this makes it much more manageable. The key is getting a handle on where things are and what they do for you. Then when a problem arises, you know where to go for help.

FILE LOCATIONS

Before going into detail on the various types of files associated with the Oracle system, let's step back and provide an overview of how these files are arranged on the disk drives of your system. If you have Oracle 6, it is hard to predict precisely where files are stored. Most installations did not have sound guidance and used whatever the DBA at the time preferred. Starting with Oracle 7, Oracle has issued guidance as to how to lay out your files in what they refer to as the Optimal Flexible Architecture (OFA). You may wish to do things slightly differently than the OFA would. Note, too, that the configuration of the Personal Oracle7 product that runs under Microsoft Windows is slightly different from those of the other products.

The Optimal Flexible Architecture was designed by the Oracle consulting staff for installations that it performed for customers. Oracle liked it so much it presents it in its Installation and Configuration Guide as a recommendation. This architecture is also the default for its Oracle installer scripts on most computer systems. Figure 8.1 shows the configuration of Oracle 7 for UNIX. This figure also shows the general concepts of the OFA, although the names of the directories may vary on other operating systems.

Figure 8.1.
The Optimal Flexible
Architecture (OFA) for
UNIX.



The starting point for this architecture is known as the Oracle base. There are three key directories under Oracle base. Admin stores log files and other information for the DBA. Data is designed to store data files. Product stores the Oracle application software and the configuration files for the Oracle home instances. The other two directories are less frequently used, at least in my experience: TAR is designed to hold tape downloads, and Local is designed for items you create and may want to store with the Oracle software.

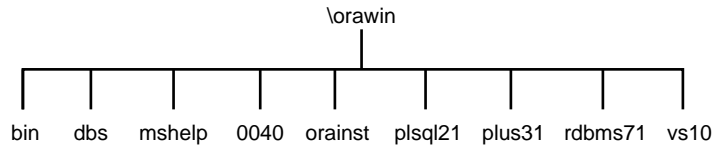
This architecture has a place for everything that you will find in a typical Oracle database. As with most standards, once you learn where things are in it, you can find those same things in any other Oracle databases. The large number of directories helps in that each lower-level directory has a specific focus and limited number of files that you have to sort through (from a few files to a few dozen if you keep things cleaned out properly). There is a single directory that contains all the executable files that you will typically need. This makes it easy to set your path so that you can access the Oracle applications. Finally, this architecture is designed to easily accommodate having multiple versions of the RDBMS software on the system at the same time. This is especially useful when converting over instances. You can bring up the new release of the RDBMS on your test database to see how it works while maintaining production on the existing release.

Very large installations usually alter the storage of data files to maximize performance at some point. Configuration alternatives are explored more thoroughly in the installation planning discussions in Chapter 17. For now, just try to get a good feel for the types of data files that you have. Each type of file will be discussed to provide you with an understanding of what it can do for you.

In almost all PC-based installations, creating a number of directories that can be split across multiple disk drives is not worth the effort. Most PCs have a single disk

drive, and most DOS/Windows users are used to having software products installed at a single directory hierarchy located off the root directory. Part of the balancing act a product has to do when porting between platforms is maintaining the same command structure and interfaces yet adapting to the customers of the host environment. In Oracle's case, they have implemented the directory structure shown in Figure 8.2.

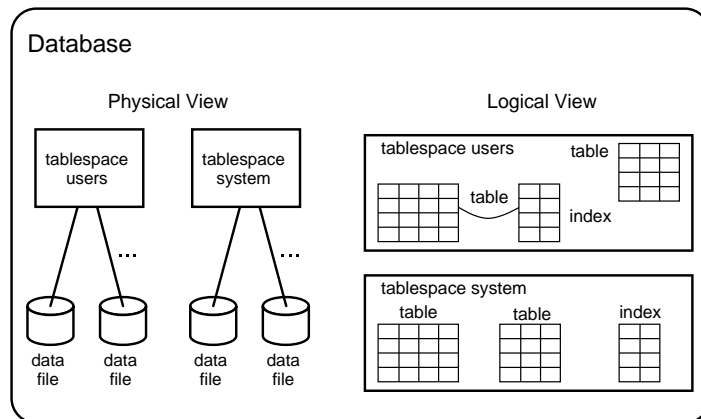
Figure 8.2.
Directory of Personal
Oracle7 under
Microsoft Windows.



DATA FILES

Data files perform perhaps the most important function in an Oracle system—storing data in a retrievable format. These files cannot be read directly from operating system utilities such as `more` in UNIX or `notepad` under Microsoft Windows. You access their contents via SQL queries, which brings up an important point. You have little control over where things are stored within a given tablespace or data file. Instead, when working with SQL commands, you specify logical structures such as tablespaces and tables. The physical structures are Oracle's responsibility. Then you, as the DBA, map the two together (see Figure 8.3).

Figure 8.3.
Physical versus logical
data structures.



Remember that each data file is associated with only one tablespace. A tablespace has one or more data files. The DBA has to make a tradeoff when determining the number of data files associated with a given tablespace. One large tablespace with all the data tables is easier to administer, and you do not have a table running out of space in one tablespace when there is a large amount of space in other tablespaces

that it cannot use. However, because you cannot control the placement of a table or index within data files in a tablespace, you are unable to split the input/output load between disks. There also are situations where you would rather a user be stopped by filling up a tablespace before taking up all the space available to the system. An example of this is where you allocate space between various projects sharing a single database. It takes system resources to keep track of the large number of data files, so you usually want to keep the number of files down to a reasonable value.

Let's explore what these data files look like internally, at least at a conceptual level. Stored within every Oracle data file is a record of the latest change number (SCN in Oracle terms) that has been applied to the system. In this manner, when you restore a backup data file from tape, Oracle can determine which changes need to be applied to the file to bring it up to date. Another point to remember about data files is that they are preallocated by the operating system to the full size that you specify in the creation command. If you allocate a file for a tablespace to 100M, the data file takes up 100M even though it contains no tables or data. This preallocation concept applies to the database objects, such as tables that you create within the tablespace, and their data files. If you create a table with an initial extent of 10M, there will be a 10M section of the tablespace reserved for that table even though it does not contain any rows of data.

The following summarizes other facts that you should know about your data files:

- ♦ There are two general types of data files—*raw* and *cooked* (don't you just love these computer terms). The normal operating system files that you work with are considered to be cooked because they use the file system utilities provided by your operating system. In some older operating systems, especially older UNIX implementations, the overhead associated with the operating system file management utilities slowed Oracle down unnecessarily, so Oracle enhanced its software to talk directly to the disk drives. These were referred to as raw disks. With most modern operating system implementations, the speed difference between raw and cooked disk drives is small and there are cases wherein the cooked disk drives can provide better performance.
- ♦ One of the primary benefits of having a number of tablespaces is to balance the input/output load across several disk drives or disk controllers. Generally speaking, you want to split tables and their indexes into separate tablespaces located on separate disks to maximize performance (at least in larger Oracle instances). You also want to separate the rollback and temporary segments (see Chapter 10) from the table and index data files in order to prevent input/output to these tablespaces from competing with each other.

- ◆ Because every write transaction to a data file is mirrored with a transaction to a redo log file, you usually want to locate your data files on separate disk drives from those of your redo log files.
- ◆ Finally, Oracle 7.1 provides an interesting new feature—the read-only tablespace. Tablespaces designated read-only are assumed to be up to date without having to check their latest update number (the SCN). This finally enables you to place tablespaces on such devices as CD-ROMs (to which you could not write to record SCN updates). It also can save time for tablespaces that contain reference data that you do not update. If one of these data files is lost, you copy the data file from the backup tape and bring it directly on line. Oracle understands that it does not have to try to apply redo and archive log transactions to bring the file up to date.

This information should start you on the way to understanding Oracle data files. Later chapters discuss the objects that you place within a data file and maintenance of the tablespaces and data file. For now, ensure that you understand what these data files are and their relationship with tablespaces. The following is the SQL query that enables you to determine the location of your data files:

```
SQL> select * from sys.dba_data_files;
```

FILE_NAME				
FILE_ID	TABLESPACE_NAME	BYTES	BLOCKS	STATUS
C:\ORAWIN\DBS\wdbrrbs.ora	3 ROLLBACK_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbtemp.ora	4 TEMPORARY_DATA	2097152	1024	AVAILABLE
C:\ORAWIN\DBS\wdbuser.ora	2 USER_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbsys.ora	1 SYSTEM	10485760	5120	AVAILABLE

```
SQL>
```

REDO LOG FILES

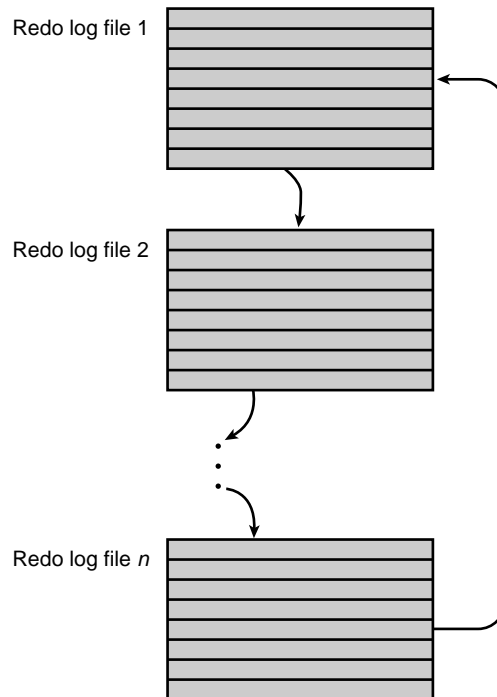
Redo log files are often referred to as the online redo log files to distinguish them from the archive log files (covered in the next section). You can control the number and size of the redo log files for your system. Basically, having a small number of redo log files that are larger can be beneficial on log switches when you are archiving the log files. This spreads out the times when the redo logs are copied to archive logs. This is important when input/output transfer rates to the disks on which the archive logs files are being created is limited. (You have to complete the copy operation before the online redo log can be overwritten.)

You have a fixed number of online redo log files that are used in a cyclical manner (see Figure 8.4). When you finish writing to the first file, further transactions are written to the second file. This process continues until you finish writing to the last file in the sequence, at which time you begin to write to the first file again. You have to be careful when you are operating in archive log mode, because you cannot overwrite the contents of an online redo log file until that file has been completely copied to an archive log file. This is important. If Oracle cannot write a redo log file to an archive log file because, for example, the file system for the archive log files is full or offline, Oracle cannot accept transactions. It sticks, giving error messages to users who try to perform updates, deletes, or insertions. Worse still, it sometimes gets so confused in this mode that it is very difficult to get things moving again. Trust me, I've had it happen several times.

Warning

Carefully monitor the file system where the archive log files are being written to prevent locking up Oracle's capability of accepting transactions.

Figure 8.4.
Online redo log file
recycling.



This setup, where one redo log file at a time records every transaction made to the system, can turn into an input/output bottleneck. Starting with Oracle 7, you have the option of creating groups of redo log files. If one of the members of the group is

on a disk that is busy, the transaction is written to an available redo log file in that group and the busy log file is synched-up at a later time. You can have several members of each of several groups of log files.

As with most of the topics in this book, there is more information to cover than space available. Appendix A shows the commands that can be used to add or delete these files. The following is an example of the command you use to determine what the redo log files are associated with in your instance:

```
SQL> select * from v$logfile;
```

```

      GROUP# STATUS
-----
MEMBER
-----
          2
C:\ORAWIN\DBS\wdblog2.ora

          1
C:\ORAWIN\DBS\wdblog1.ora

SQL>
```

ARCHIVE LOG FILES

From the last section, you should understand that archive log files are merely sequential copies of the online redo log files. Their contents are a series of transactions that have been applied to your database. What else do you need to know about archive log files? Two key points: the options that are available for storing archive log files and how to set up the database to use archive log files.

Archive log files have two basic storage options. The first option is to use space on one of your disk drives. The other option is to write these log files directly to magnetic tape. There are pros and cons to either approach:

- ◆ Writing archive logs to tape saves the much more expensive disk storage space for other purposes.
- ◆ Archive logs written to tape have to be recovered from the slower tape medium in the event you need to do a recovery.
- ◆ Writing archive logs to tape requires a dedicated tape drive, which is not available on all systems.

Getting Oracle to start writing archive logs is a somewhat more difficult subject. When you create a data file within a tablespace, Oracle writes to it automatically when users place data into that tablespace. Oracle automatically writes to the redo logs that are created when you create your instance. However, you, the DBA, have to go through a somewhat redundant and extremely sensitive process to get Oracle to write archive logs:

1. Go into the initialization files and set the `LOG_ARCHIVE_START` parameter to `True`. Don't be fooled; this parameter doesn't really start archive logging as you would expect. It sort of gives Oracle permission to start archive logging if all the other parameters are set up and only takes effect when you restart your Oracle instance.
2. Tell it the format for the name of your archive log files. The exact syntax can be found in Chapter 17 of the *Oracle 7 Server Administrator Guide* (chapter numbers may vary in different releases), but you often can accept the default format.
3. Tell Oracle where to stick the files and how to name what you are about to start archiving. This entails specifying a directory path and the first few characters of the archive log files themselves (it is not just a directory path). An example might be to have a series of archive log files that begins with the text string `alog` located in the `/disk57/oracle/logs` directory. Set the `LOG_ARCHIVE_DEST` parameter to the following:

```
LOG_ARCHIVE_DEST = /disk57/oracle/logs/alog
```

The second part of the archive file name is a variable string of characters that typically include sequential numbers. This format is specified by the `LOG_ARCHIVE_FORMAT` parameter. There are a number of options for this parameter, but the default usually works well and looks something like:

```
LOG_ARCHIVE_FORMAT = '%s.arc'
```

4. Shut down (normally) your Oracle instance, perform a complete backup (for safety's sake, please don't skip this step), and bring up the instance with the database not open (that is, just give `SQL*DBA` the startup mount exclusive command). When the database has been started, enter the following command:

```
alter database archivelog;
```

5. You can now open up the database. Be sure to verify that archive log files are actually being created. This is a very sensitive procedure and it is ripe with opportunities to make little mistakes in syntax that will cause you problems. You can verify that log files are being written by manually forcing a redo log file switch with this command:

```
alter system switch logfile
```

You may have guessed by now that archive log files have bedeviled me on several occasions. I usually wind up grouching when Oracle rejects my `LOG_ARCHIVE_DEST` parameter for some little syntax error. It is a tedious procedure. Once you have archive logging operational, you always have to be concerned with running out of space for archive log files and locking up Oracle's capability of writing transactions to the database. However, for the comfort of knowing that you can recover all data committed up to the point of failure of a disk drive, I guess it's worth it.

CONTROL FILES

After that rather invigorating discussion of the archive log files, it is time to cool off a bit and discuss a relatively easy topic—the control files. If you are lucky, the only time that you have to bother with control files is when you create an Oracle instance. This is because control files are used by Oracle internally. You cannot manually edit them, you cannot control their size, and you can move them if you have to. Otherwise, the only time you will notice them is if one is missing. (Oracle will give you an error message indicating which control file is missing or damaged.) When this happens, you copy one of the other control files (they are identical to one another) into the location of the missing control file and restart the system.

Control files are a road map of the database as it is physically laid out on your computer system. They store such tidbits as the name of the database, the data files and redo log names, and a record of when they were created. You can add control files and delete existing ones if you want (see Appendix A for the commands). One rule to remember when locating control files is that you always want at least one control file available so that you can re-create the others. You can re-create the control files manually in Oracle 7, but you do not want to have to try this. Therefore, spread your control files out over several disk drives whenever possible so that you always have one to start your instance.

INITIALIZATION FILES

Oracle determines the physical configuration of an instance by accessing the control files. However, there are a number of configuration parameters related to tuning that you do not want to reset manually each time you start an instance. To make this easier, Oracle provides a series of initialization files that are similar to the `autoexec.bat` and `config.sys` of Microsoft DOS. On multi-user Oracle systems, there are two primary initialization files that are used:

- ◆ `init.ora`. This is the main file. The actual filename contains the Oracle instance's SID (system ID), so that an instance named `blue` would have an initialization file called `initblue.ora`.
- ◆ `config.ora`. Typically, `config.ora` files contain the locations for the control, archive, and dump files. They also contain the database name and the database block size parameters. Oracle intended these files to contain values that have parameters that are the same between instances; `init.ora` files contain parameters that are tuned and vary between instances. As with `init.ora` files, the `config.ora` files contain the SID in their name (for example, `configblue.ora`). The `config.ora` file is called by placing a line in the `init.ora` file that says to use a second initialization file (`ifile = filename`).

The initialization files are simple text files that you can edit with standard operating system editors, such as the wonderful vi editor in UNIX or notepad under Microsoft Windows. The format is simple; you set a variable equal to a value. I have two gripes about the setup of these files. First, I hate to have to keep two files edited when one can do the job (personal preference). Second, the data is so scattered in these files with so many options commented out that it is difficult to see what you are setting and you have no history of what you were using in the past. Therefore, I like to redo my initialization file to contain the following sections (a sample init.ora file for UNIX systems is contained on the disk):

- ◆ A header that contains the name of the file, its purpose, and a history of all parameter changes (so that if a parameter change does not work out, you know where to set it).
- ◆ A grouping of all parameters that affect the size of the SGA. I do not keep commented-out parameters, only live ones.
- ◆ A grouping of all parameters that affect this size of the PGA.
- ◆ A grouping of all parameters that deal with logging.
- ◆ A grouping of the parameters associated with national language support (NLS), which is Oracle's tool to support databases that use multiple different languages for the data and such things as date formats.
- ◆ A grouping of all parameters associated with the multi-threaded server, if that option is used.
- ◆ A general category that contains the other miscellaneous parameters, such as whether the multiple data base writer processes are used.
- ◆ Finally, in certain cases Oracle will suggest you add some undocumented parameter to your system to make it run better. This is often the case when you are working with a large data warehouse. I like to keep these parameters in a separate section so that I know where they originated.

Before moving on to the next section on log and trace files, you need a few more notes. In multi-user Oracle systems, you can find the init.ora files in the dbs subdirectory under the ORACLE_HOME directory (the directory under which you locate all your Oracle software and administrative files. If this does not contain the actual file, it will contain at least a link that points to the init.ora file. The config.ora files are located in the pfile subdirectory under the ORACLE_BASE/admin/*instance_name* directory. In Personal Oracle7, there is only an init.ora file, located under the /orawin/rdbms71 directory. This is probably enough to get you started in the world of Oracle initialization files. If you have a test Oracle instance available to you, try altering the configuration files and tuning parameters in this instance before you work on the instance that your company is depending on for its daily operations. If you do not have this luxury, be careful, make a copy of the original initialization files

with another name (for example, `initblue.ora.1jan95`), and then try adjusting one parameter at a time until you build up your confidence.

LOG AND TRACE FILES

Recall that Oracle really wants to help you out when a problem occurs. Therefore, it takes a few CPU cycles in its utilities to write out a record of important events that occur to a set of log files. The log file that records major events in the RDBMS itself is the alert log file. As always, the exact name and location of this file is a little more complicated than just `alert_log`. The actual file contains the name of the instance with which the events are associated. For our near-famous blue Oracle instance, the alert log file would be called `alert_blue.log`. The next logical question is what exactly gets placed in the alert log:

- ◆ Major DBA activities, such as starting or stopping the instance or issuing commands that create, modify, or drop the database, its tablespaces, or rollback segments
- ◆ Internal Oracle errors
- ◆ Messages related to multi-threaded server problems
- ◆ Errors related to automatic refreshes of a snapshot

The trace files are produced by the Oracle background processes when they sense that they have a major problem or when one of the processes detects that another background process is in trouble or missing. The content of the message varies depending on how much information the process writing the message can sense. It usually contains the date and time of the problem, with which background processes the problems occurred, numbered Oracle error messages that you can use to discuss the problem with Oracle, and some explanatory text. If you have a problem with Oracle, save these messages until after you have determined the exact cause of the problem and have a fix for it. Some versions of Oracle write trace files every time you start. This does not indicate errors; it's just logging a successful start.

One common Oracle feature that does not store its logging information in the alert log file is the SQL*Net process. Instead, you will find a log file under the appropriate SQL*Net directory related to the protocol you are using for your system (TCP/IP, for example). This makes some sense in that alert logs are tied to a specific instance, but a single SQL*Net process can service multiple instances.

Now that you are enthusiastic about the wealth of information that is available to you in the log and trace files, you probably can't wait to learn where to find them. The answer is that the location where these log files are kept is a parameter specified in your initialization files (specifically the `config.ora` for the instance in question). That parameter is called `BACKGROUND_DUMP_DEST` (the destination for the dump files of the Oracle background processes, which is typically `ORACLE_BASE/admin/instance_name/bdump`).

One final topic to consider is cleaning up after yourself. The log files just keep growing and growing as more events are recorded. The trace files sit out there until you do something with them, eating up more and more disk space. Therefore, it is a good idea to implement housekeeping procedures wherein you regularly clear out the log, core, and trace files. My favorite option is to set up a script that automatically copies the alert and SQL*Net log files to files that have the date that they created as part of their name (for example, alert_blue.log.013095). Then purge all the trace and log files in the appropriate directories that are over 30 days old. The system stays clean and you never have to scan through a 1M log file to look for data on a problem that you encounter.

SUMMARY

This completes the discussion of the second major part of the Oracle database instance—the files. This chapter reviewed these files with you at a level that enables you to be a productive DBA. You explored the problems that you encounter with certain types of files (archive logs, for example) and learned where to find these files. You should be getting fairly comfortable about the types of files that Oracle uses and starting to see how the overall architecture is fitting together. The next chapter discusses the third piece in this puzzle—the operating system processes that Oracle uses to move data between memory areas and disk files in order to service user requests.

- 
- The System Monitor
 - The Process Monitor
 - The Database Writer
 - The Log Writer
 - The Archiver
 - The Recoverer
 - The Lock Writer
 - Dedicated Server Processes
 - Multi-Threaded Server Processes
 - SQL*Net Listeners
 - Parallel Query Processes

CHAPTER 9

Oracle Processes

This is one more chapter on the theory of how Oracle is working behind the scenes for you and this one presents some of the practical topics that you have been waiting for, such as tables, tablespaces, and privileges. Theory is difficult for some practical-minded individuals, but with a system as complex as Oracle, you need to understand what is going on to deal with problems that arise.

This chapter presents an overview of the Oracle processes and how they work together first. Then you explore each of the major background processes and learn just enough detail to give you a sound understanding of what the process does and with what it interfaces. Finally, you see how Oracle implements all these functions in the Personal Oracle7 product.

OVERVIEW

Refer to Figure 6.2 in Chapter 6. This is a simplified drawing, which this chapter expands on to cover more processes. For now though, it illustrates the basic concepts that you should understand. The Oracle background processes serve four basic types of functions:

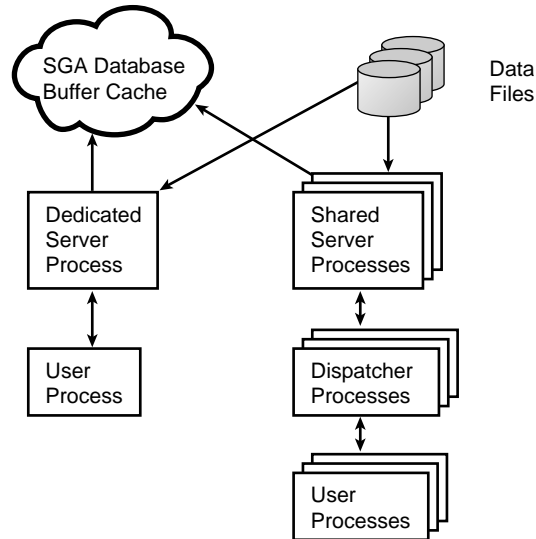
- ♦ *User service processes* provide connections between the user's individual application processes (a SQL*Forms session) and the memory areas of the Oracle database. This group includes processes such as the SQL*Net Listeners, the Multi-Threaded Server processes, parallel query processes, and the Lock Writer.
- ♦ *Data writing processes* take data that is stored in the database buffer discussed in Chapter 7 and write it to the data files discussed in Chapter 8.
- ♦ *Log writing processes* take the data stored in the log buffer and transfer it to the redo log files. The archive log writer transfers data from the online redo log files to the archive log files.
- ♦ Finally, *monitoring processes* keep an eye on things to ensure that processes are functioning correctly. These processes are responsible for writing many of the trace files discussed in the last chapter.

It is time to move beyond this simple picture and dig into the technical details. The first task is to map these conceptual processes into the actual set of processes that you would see in a process listing on the host computer.

The first set of processes are the user service processes. These are your links to the Oracle system. There are two basic options when it comes to connecting users to the Oracle memory areas (see Figure 9.1). The traditional method spawns a separate

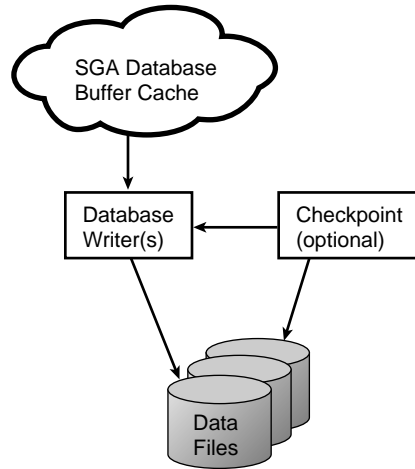
Oracle server process for each user process that needs attention. Oracle refers to this as a dedicated server process. This worked well under Oracle 6 wherein the databases and the number of users were relatively small. However, as businesses moved toward Oracle and UNIX computers, systems that supported a much larger number of users were developed. It became impractical to devote an entire process to each of the users on the system. This is especially true when you consider that most of the time the users are not actively querying the database. (We take a very long time to type in queries and read results from a computer's point of view.) So Oracle developed the multi-threaded server (MTS for you acronym fans). With the multi-threaded server, the users go to a dispatcher who assigns their requests to one member of a pool of server processes. This way, Oracle needs to allocate only enough server processes to handle the number of users who are concurrently communicating with the data base.

Figure 9.1.
User service processes
in Oracle.



Next on our agenda is a discussion of the set of processes that write data to the data files. There are two types of processes in this category (see Figure 9.2). The first is the one that you would expect from earlier discussions, the database writers (which have DBWR in their titles). There is an option to consider with these processes. Oracle 7 enables you to put multiple database writers to work for you. With this option, you can have one database writer at work while another is bottled up waiting for input/output on a particularly busy disk.

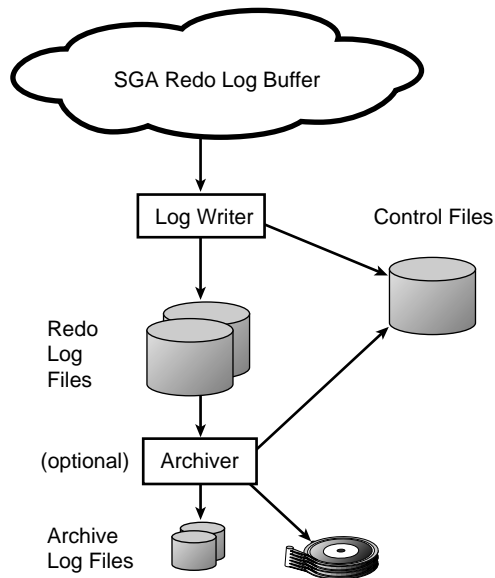
Figure 9.2.
Processes that write to
the data files.



The second data file writing process is not one to which you normally give any thought. Recall from the last chapter that each of the data files contains the number of the latest transaction that has been applied to it. Something has to remember to write that number to each of the data files on the system. Normally, this work is done by the log writer process in its spare time. However, on systems wherein there are a large number of data files, the checkpoint function (which is what Oracle calls writing the SCN to each of the data files) can slow down the performance of the log writer. To keep the log writer focused on its primary job, you can start a separate checkpoint process to handle this task (which has CKPT in its name). This is controlled by the `CHECKPOINT_PROCESS` parameter in the initialization files. I have never used it, but those working on large transaction processing systems should at least consider it.

The third process group is the log writer group. Recall from the last chapter that there are two types of log files (online redo and archive), so it makes sense that there would be two log writing processes (see Figure 9.3). The log writer process (LGWR) is responsible for transferring transactions that have occurred from the redo log buffer in the SGA to the online redo log files. It is responsible for keeping track of which log file is ready to receive data. It also is responsible for ensuring that the log file has been copied to an archive log file before it is overwritten when the data base is operating in archive log mode. Finally, if there is not a separate checkpoint process running, the log writer is assigned the additional work of recording the latest change number processed by the system into each of the data files.

Figure 9.3.
The Oracle log writing
processes.



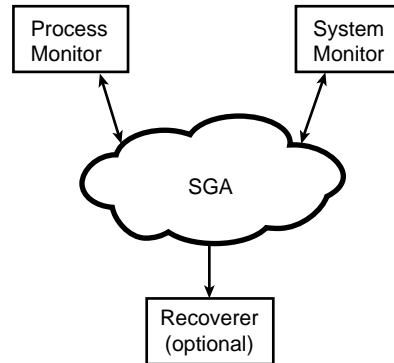
The second log writing process is the archiver (ARCH). Its duty in life is to make copies of the online redo logs to either tape or disk when the log writer process gets through with them. As mentioned earlier, if this process gets stuck (such as when a tape runs out or a disk file system fills up), it can hold up all transactions to your Oracle system. This process is started when you issue the `alter system archive log start` command. To start this processes automatically on future instance startups, set the `LOG_ARCHIVE_START` parameter to `TRUE` in your initialization file.

The monitoring processes are the final group that needs to be discussed. There are three processes in this group (see Figure 9.4). The first of these processes is the system monitor (SMON). The system monitor is assigned the task of cleaning up the instance at startup and whenever it is needed during operations. The next monitoring process is the process monitor (PMON). This process focuses on cleaning up after user processes that fail. Finally, the recoverer is a very highly specialized process that focuses on problems with distributed database transactions. If you are not using the distributed database option, you will never see the recoverer.

Now is probably a good time to discuss how these background processes work. On most UNIX and VMS systems, you find these processes out there whenever the instance is started. The database does not have to be mounted (associated with a set of Oracle instance processes) or opened (made available to general users). The

processes are started when you start the instance. It would not make any sense to have these processes continuously hitting the CPU. If you are using a system primarily for queries, you do not need the database writer very often, so why load down your expensive CPUs?

Figure 9.4.
The Oracle monitoring processes.



Oracle has designed the software for these processes to be activated under two circumstances. First, they are activated by other processes when needed. For example, the system monitor is activated during startup to see whether the instance needs to be recovered. Otherwise, these processes sleep in the background and wake up every so often to see whether their services are needed. If they are needed, they perform their work and then go back to sleep. If they are not needed, they go to sleep immediately. Those of you who were worried that the large number of processes that Oracle was spawning would load down your CPU can relax. Unless you find a bug in a new release of the Oracle software, these processes consume substantial amounts of CPU time only when they are needed.

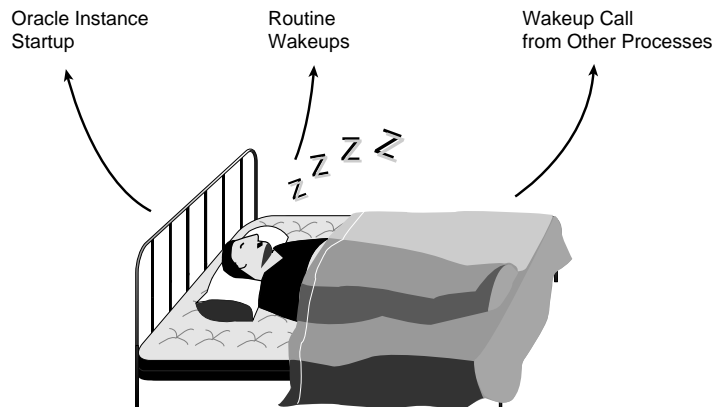
These are the processes with which most users are concerned. Many of the applications that are based on Oracle have their own series of background processes to watch. At this point, you should be comfortable with the idea of several background processes that need to be present when an Oracle instance is started. They are specialists, each focused on a particular set of tasks. The next sections explore each of the individual processes again, emphasizing additional features and controls.

THE SYSTEM MONITOR

The system monitor's (SMON) assignment is to clean up after the Oracle instance itself. Just as the process monitor focuses on cleaning up after user processes, SMON is the janitor for the Oracle background processes and data files. The system monitor

is activated under three circumstances (see Figure 9.5). First, it checks at startup to see whether the instance needs to be recovered. If the instance was not shut down cleanly (shutdown normal), there may be transactions that were in process that need to be rolled back, temporary segments (places where Oracle stores data in the data files that are not permanently committed) for such things as sorts that were being used that need to be cleared out, and so forth.

Figure 9.5.
The Oracle system monitor (SMON).



The system monitor is also designed to wake up routinely to see whether the Oracle instance needs some cleanup. SMON checks to see whether there are any temporary segments that are no longer needed and that could be cleaned up. This is important to ensure that the maximum amount of temporary space is available for new requests. SMON also has the task of searching for ways to group together the free extents in the data files. If you ask for a 1M extent for a table, Oracle searches for a 1M contiguous extent (all the blocks are located next to one another for speed of access). You could find yourself in a situation where you have several megabytes of free space within a data file, but they are scattered in a large number of small chunks. SMON will try to move things around so that you have larger free chunks of disk. However, its capabilities are limited, so you may occasionally have to compress extents within a tablespace, which is covered in later chapters.

The system monitor can also be called by other processes. If the database writer detects the need for temporary segments, it can ask the system monitor to see what it can do to free some up, for example. This is an important feature to note with Oracle. The processes have been designed to work together to give you the maximum chance to keep the instance operational without returning errors to the users. This is not perfect and you will still receive errors and rare crashes. However, that's why there are still jobs for the database administrators.

THE PROCESS MONITOR

The process monitor (PMON) devotes its time to cleaning up after user processes. It wakes up routinely and can also be called by other processes when they detect the need for its services. The following are some examples of tasks that PMON may have to perform when a user process dies:

- ♦ Removes the process ID
- ♦ Cleans up the active transaction table
- ♦ Releases any locks that the process may have
- ♦ Removes items in cache that the process was using

In the multi-threaded server environment, PMON is responsible for restarting any dispatcher or server processes that have died. The multi-threaded server has a minimum and a maximum number of processes that can be active. The actual number of processes is determined by user needs, up to the maximum allowed. Here PMON intervenes only if the number of multi-threaded server processes is less than the minimum specified by the DBA in the initialization files.

THE DATABASE WRITER

The database writer (DBWR) process is responsible for transferring data that has been modified in the buffer cache memory area to the appropriate disk data files. Its goal in life is to ensure that there are always free buffers waiting in memory when a user process wants to write a modified record to the database buffer cache. To review our buffer cache theory from a few chapters back, whenever a user adds or modifies a record from a table, the modified record is written to the buffer cache. Oracle uses the term “dirty” to classify that buffer in memory to signal that it cannot be overwritten until it has been transferred to disk. You have only a fixed number of buffers in memory (determined by the size of the database buffer cache that you establish in your initialization file). Every time you dirty a buffer it reduces the number of free buffers (available for writing). If the total number of free buffers becomes too low, there are none available for user processes to place the data they obtain on queries (remember, you read multiple records one at a time and store the ones that you do not immediately need in the buffer cache). This is where DBWR gets called into action.

The database writer is one of the enforcers of the least recently used algorithm, the other being the user processes that may overwrite non-dirty buffers that have not been used recently. Many records can be used and updated frequently. The database writer avoids writing records to disk that are likely to be needed soon (as determined by their frequent usage in the past). A classic example of relatively inactive records is order data on an order-entry system. Typically, an order is entered and not

accessed again until someone is ready to ship that product. These types of records are quickly removed from the buffer cache to make room for other orders. A type of record that stays in the buffer cache is frequently used lists of legal values that are scanned every time an order is entered to validate that it is correct.

So far, you have learned what the database writer does. You also have seen how the database writer determines which records to write to disk from the buffer cache. The next piece in this puzzle is to determine when the database writer does its writing. There are four conditions under which DBWR is called into action:

- ◆ A server process writes a new dirty buffer into the database buffer cache. It is smart enough to detect that the number of dirty buffers has exceeded a limit that the DBA has set for dirty buffers (one-half of `DB_BLOCK_WRITE_BATCH`, which Oracle calculates based on your database buffer size). Being a good citizen in the Oracle background process world, it politely informs DBWR that it is time to wake up and do some writing.
- ◆ A server process is looking for a free buffer to write to and cannot find it. It does not scan every possible buffer in the buffer cache because this takes too much time. Instead, it searches a number of buffers specified by a parameter set by you, the DBA (`DB_BLOCK_MAX_SCAN_CNT`, which is calculated by Oracle based on your database buffer size). When it has looked at this many buffers and has not found a free one, it concludes that it is time for some writing (there are a lot of dirty buffers out there). It then signals DBWR to begin writing.
- ◆ The database writer wakes itself up routinely to see whether there is any work for it. This period is set to three seconds.
- ◆ Finally, the log writer or the checkpoint process needs to have those records that show the latest change number applied to the database written to each of the data files when a checkpoint occurs. Because this function is especially important for database recovery purposes, these processes get special attention and signal the database writer to write these records when they are ready.

How much does the database writer transfer to disk when it is called into action? The checkpoint records are directed for immediate write when the database writer is awakened for a checkpoint. For the other situations, the number of records written depends on how DBWR was awakened and certain Oracle configuration parameters.

If the database writer is awakened by user processes that detect that the number of dirty buffers is too high or cannot find a free buffer, the database writer goes to the list of dirty buffers and writes a batch of records. There may be times when there are not enough buffers available on the list of dirty buffers that Oracle maintains (there is a time delay between dirtying the buffer and this list being updated). If

DBWR cannot find enough data on the dirty list to fill up its transfer quota, it scans the list of buffers, starting at the least recently used end, to find dirty buffers to write to disk. The database writer is a very service-oriented process.

If the database writer is awakened on a timeout, it uses a slightly different approach to writing data buffers. It goes through the list of buffers, starting at the least recently used end and selects dirty buffers to write to disk. It searches twice as many buffers as was specified by the `DB_BLOCK_WRITE_BATCH` parameter on each awakening (remember, many of these buffers will not be dirty and therefore not need writing). Any dirty buffers it finds are written to disk. Each time DBWR wakes up it searches a new set of buffers on the least recently used list of buffers. Eventually, if the database has a relatively light data entry rate, all the dirty buffers in memory will be written to disk. This is the case in most of the systems that I have worked with and would probably apply to all but the busiest transaction processing systems.

As mentioned earlier, certain operating systems enable you to activate multiple database writer processes to share the load. The number of database writer processes that you use is controlled by the `DB_WRITERS` parameter in the `init.ora` file. Oracle recommends having at least one database writer process for each disk drive in your system. We did not implement 135 database writer processes for the one large information warehouse I worked on that had 135 mirrored disk drives (2G each). I would recommend having enough database writer processes for the expected number of disk drives that would be written to at a given time. Start with a value that you consider reasonable and then increase its value and see whether your performance improves.

Now is as good a time as any to bring out one of the fundamental design concepts used by Oracle to improve performance in its processes—parallelization. This is especially true in Oracle 7.1 where you can even assign multiple processes to service a single query. Oracle enables you to assign multiple database writers to do the work when it becomes too much for one to handle. Oracle enables a separate checkpoint process to take some burden off the log writer. You need to understand your computer's architecture before you jump at the chance to improve performance by adding additional processes. The parallelization techniques are useful for systems with a number of disk drives (the parallel database writers) and computer processors (as in the parallel query option). Certain computers, such as most of the HP and IBM UNIX servers, are based around a single processor architecture. They would not benefit from the multiple query processors as much as a large multiprocessor-based computer such as the Sequent and Pyramid families. Computers with all their Oracle data stored on one or two disk drives also would not benefit from a large number of database writer processes. Just ensure that you understand what you are trying to do when you add parallel processes and that this will work with your computer architecture.

THE LOG WRITER

Recall from the past couple of chapters that Oracle writes every committed transaction in two places to promote its capability of recovering in case a data file is completely destroyed. The first location, the data file itself, was written to by the database writer process covered in the previous section. This section deals with writing the other record of a transaction from the redo log buffer to the redo log files. This function is performed by the log writer background process (LGWR).

The general concept of the log writer is simple. Every transaction made to the database is recorded in the log buffer. This buffer operates on a first-in, first-out basis. There are multiple redo logs in the system and the log writer writes to them in a cyclical manner. A slight complication to consider is the fact that users can enter transactions on the system and wait for a while until they commit them. Until the user (or the software package) issues a commit statement, these records are not official and can be backed out without any changes to the data files. Normally, commits occur frequently and you do not have a large number of uncommitted transactions hanging around. Sometimes you do have a lot of uncommitted transactions. The log writer is designed to deal with this situation and makes special exceptions for handling commit statements received. Therefore, the processing rules for the log writer process are the following:

- ◆ The log writer wakes up and performs a write when a commit statement is received from a user process.
- ◆ The log writer wakes up and writes redo log buffers every three seconds. The processes that read the logs on a recovery are smart enough not to record changes in the data files until that commit statement is received for that particular transaction.
- ◆ The log writer writes redo log buffers when the redo log buffer becomes one-third full.
- ◆ Finally, when the database writer process writes records in the database buffer to disk, the log writer ensures that the corresponding redo log buffers are written to the redo logs.

Of course there are some fine points to this process. For example, if there are a lot of commits occurring in your database, Oracle may choose to transfer groups of commit records to the log files for efficiency. Recall also that you can actually have several identical members in a group of redo log files. That way, if one of the members of the group is not available (say the disk is busy with other input/output operations), the writes can occur to an available member of that group. Finally, under normal circumstances where there is not a separate checkpoint process, the log writer is responsible for writing records to the database buffer that record the

latest change number (SCN) applied to the database in the headers of all data files. The database writer then transfers these records from the database buffer to the disk files.

THE ARCHIVER

The archiver (ARCH) is a conceptually simple process. When the log writer finishes filling up one of the online redo log files, it signals the archiver to copy it to the next archive log file in the sequence. The archiver sends the copy of the online redo log file to either a new file on a disk drive or to a tape that you have designated. It can, of course, do its job only if there is room available on the tape or the file system on which you tell it to write the archive log file. The archiver process is started by the log writer only when the database is operated in archive log mode (that is, the DBA issues an `ALTER SYSTEM ARCHIVE LOG START` command or you start your instance with `LOG_ARCHIVE_START = true` in the initialization file).

THE RECOVERER

Another process that is present only when you use an optional database mode is the recoverer (RECO). This process provides service when you have chosen to implement a distributed database (discussed in Chapter 4). The purpose of the recoverer is to deal with problems that occur during distributed transactions. There are special rules that have to be applied when you are trying to deal with replicated data tables or other remote tables. Many problems can occur when wide area networks are used. The basic transaction writing processes are trained to yell immediately if there is a problem writing to the local disk drives. Oracle has designed the recoverer to be a lot more tolerant of such problems. It waits until connection is reestablished with the remote database and then goes over the transactions that it is not sure were recorded in the remote system. It resolves whether there were any updates made from other remote systems (you can link many different computers with different Oracle instances via the distributed option).

THE LOCK WRITER

The next specialist process is the lock writer (LCK). This process is present only when the parallel server option is being used. In this situation, you cannot use the single instance locking mechanisms that are provided by the Oracle memory areas of your instance. Instead, these special lock writer processes (normally one, but you can have up to 10) talk to one another and ensure that records are properly locked for shared databases. Again, the logic for splitting out this function is that you do not want to weigh down your existing background processes with the burden of

negotiating between instances. This is a much slower process because the messaging is not occurring through the fast shared memory area mechanisms of a single instance.

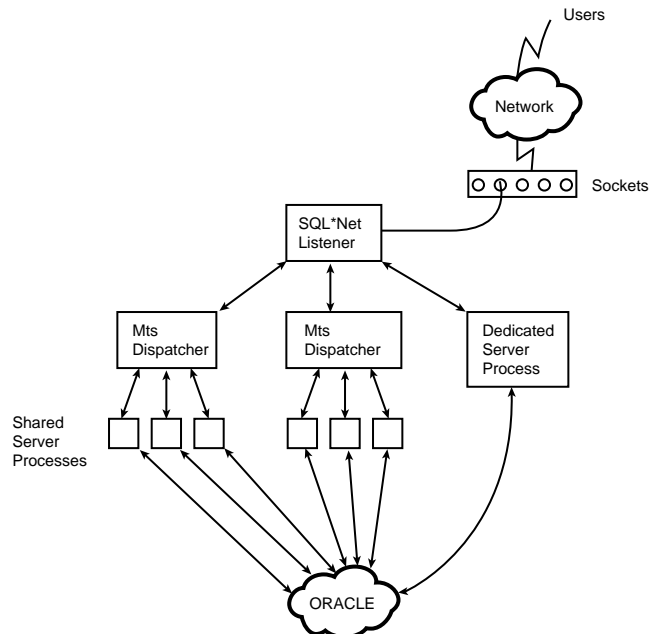
DEDICATED SERVER PROCESSES

These processes serve as the interface between a user process that is sending requests to the database and the memory areas and disks of the Oracle instance. They provide all the intelligence as to who to talk to for what service and where to find things. They have been used in Oracle since the beginning to provide this connectivity. The key to remember is that all users get a dedicated server process when they connect to the Oracle instance and it needs to remain available until they disconnect from the instance.

MULTI-THREADED SERVER PROCESSES

Starting with SQL*Net version 2, Oracle provided the multi-threaded server option. The tie with SQL*Net is logical because the large number of users accessing Oracle is typical of client-server environments. The multi-threaded server is designed to enable a number of users to share a pool of server processes (see Figure 9.6). Without this configuration, each user requires a dedicated server process as described in the previous section.

Figure 9.6.
The multi-threaded
server architecture.



Everything starts with the SQL*Net version 2 Listener (the process that monitors the network that is described in the next section). This process detects that a user wants to connect to a particular Oracle instance. If this instance is using the multi-threaded server and has a dispatcher process that is available, it transfers the requests to that dispatcher for further processing. The dispatcher determines whether the user is allowed to connect to that instance and, if so, provides that connection. When the user has a transaction or query that needs database attention, that request is routed to one of the available server processes.

A few points about the multi-threaded server are in order. First, there needs to be at least one dispatcher for each of the network protocols supported on that system (TCP/IP and IPX cannot share a dispatcher). You can have multiple dispatchers for a given protocol. You determine and set the number of dispatchers based on need. If all the dispatchers for a given protocol are busy, the listener is smart enough to create a dedicated server process for the user.

SQL*NET LISTENERS

The SQL*Net listeners are designed to provide an interface between Oracle and the low-level communications network signaling world. Basically, the network protocols have special addresses available for different types of communications. You use one on those addresses (think of them as separate telephone lines) and designate it for SQL*Net traffic only. The SQL*Net listener waits for someone to send a signal to that address. When this happens, the listener process is responsible for routing the signal to the appropriate instance and spawning a dedicated server process or providing a connection to a multi-threaded server process.

PARALLEL QUERY PROCESSES

Parallel query processes are an Oracle 7.1-specific feature. When you look at the tasks involved with complex queries, you find that there are a lot of steps that have to be performed. These tasks can be performed by a single operating system process (as is done in Oracle 7.0 and earlier). However, because most operating systems assign these tasks to a single CPU, you are limited in the total number of steps that can be executed per second and this helps to fix the total response time for the query. If you can take this big query and divide it into a series of tasks (either by dividing a large table into sections for different processes or one to do a query on the index and another to search the main table for the records that match the index search), you can assign multiple CPUs to perform the work and therefore get the job done more quickly. This is the purpose of the parallel query processes in Oracle. The DBA spawns a number of them when you operate the database in parallel query mode.

They wait until a user process issues a query that can benefit from their services. When this happens, they jump in and start performing parts of the query as determined by a special set of logic used to divide the work.

ORACLE 7 FOR MICROSOFT WINDOWS

Those of you familiar with the Microsoft Windows environment may be wondering how Oracle can implement all these background processes and get full multitasking performance. The answer is that it doesn't. You get a single process icon that sits minimized on your desktop waiting for work. The SGA is created and waits to be serviced by this process, as do the disk files. This accommodates the fact that Microsoft Windows was not really designed to be a large multitasking system (that is what Windows NT, OS/2, and perhaps the new Windows 95 products were designed to do).

It is interesting to look back at the product directions that Oracle took with regard to PCs. It had DOS-based products under version 5 of Oracle that provided services similar to those on multi-user Oracle databases. Oracle decided not to port version 7.0 of its database to DOS because it was not robust enough and it thought the market was not there. Everyone at the time was sure that OS/2 or Windows NT was going to drive DOS and Windows out of existence. Instead, the PC and Microsoft Windows database market grew sharply and people wanted to standardize on a single product for their organizations. Finally, with Personal Oracle7, Oracle acknowledged the market for PC database products and that Windows (or Windows 95) was going to be the market leader in operating systems for at least the near term.

SUMMARY

This chapter was designed to review in slightly more detail the processes that are involved with a working Oracle instance. Not every process discussed will be seen on any given Oracle instance. They are activated only when you need them. This chapter presented the level of detail that most DBAs need. If you need further detail on the parameters that control these processes, please refer to the monitoring and tuning chapters (Chapters 30 and 31) in this book or to the more detailed references in the Oracle7 Server Concepts Manual and the Oracle7 Server Administrator Guide.

Congratulations! The last four chapters covered most of the more difficult theory behind the Oracle RDBMS product. You may question why you had to learn so much theory to get the job done as a DBA, but you'll just have to trust me. Now on to some of the more practical topics that you can use daily in your work.

-

- Overview of Storage and Access
- Tables
- Indexes
- Views
- Synonyms
- Stored Procedures
- Clusters
- Sequences

CHAPTER 10

This chapter presents the database objects that you will be working with almost every day as a DBA—tables, indexes, views, and so forth. This is an important chapter to master. Although you will have time to look up the location of the alert log file, for example, when you run into a problem, you will not have time to look up what a table is each time you work with one. So let's get started.

First, consider the goals for this chapter. Some of the topics you may already know, of course, if you are an experienced developer on the system. However, a little review never hurt anyone. Basically, this chapter's goals are the following:

- ♦ Explain the logical data storage structure within Oracle.
- ♦ Explain the data storage objects (which you can think of as tools or devices) that Oracle provides.
- ♦ For each of the storage objects, discuss the implications of using the object and any limitations that Oracle imposes.
- ♦ Discuss the management considerations for these objects that a DBA should know.
- ♦ Finally, provide an introduction to how you access the information contained in these data storage objects.

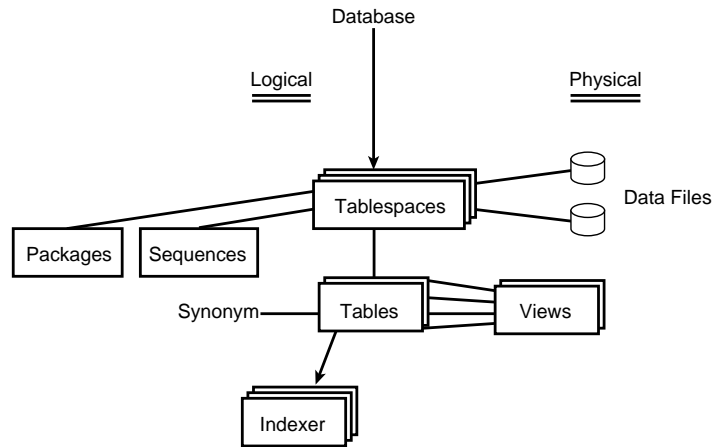
OVERVIEW OF STORAGE AND ACCESS

Chapter 8 explains the way Oracle physically stores its data—data files, log files, and so forth. However, Oracle completely isolates the users from the physical side of storage. You do not get to specify which data file to put your data in or which log file you wish to write to. Instead, Oracle provides a series of logical constructs in which you place things. These logical constructs isolate the average users from having to worry about the storage details and makes their lives much easier. The DBA is the person who is responsible for connecting these physical and logical worlds together and that is what this section is all about.

Figure 10.1 represents the contrast between the physical storage world and the logical storage world. The physical storage world can be seen in the form of disk files. The logical storage world consists of the objects that the user accesses and developers create.

At the highest level, you have the database. An Oracle instance mounts a single database, although by using SQL*Net you can access information in other databases. This database has a set of internal tables that store the data dictionary, user ID information, security tables, privileges, and all the other information needed to implement the Oracle system described in this book. A key point is that a database is complete within itself and can be mounted by another Oracle instance and it still provides the same answers to queries made to it.

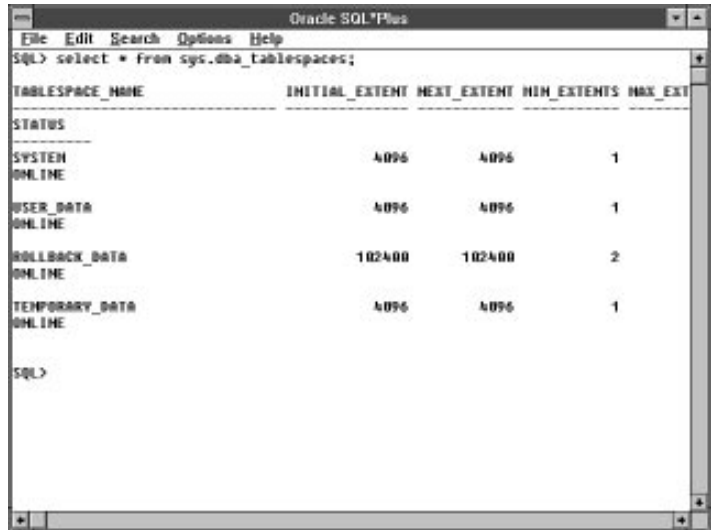
Figure 10.1.
Data storage hierarchy.



The fundamental unit of storage within a database is the *tablespace*. The tablespaces contain the various other types of objects to store user data. They also contain the series of tables, views, and so forth that Oracle itself uses to store information related to the operation of the database. The tablespace is the fundamental unit of storage for a second reason: It is the link to the physical storage world. A tablespace has one or more data files associated with it. You cannot control which data file Oracle uses to store a row that you are adding to the table, but you can control the tablespace, which at least narrows down the list of data files that you will be accessing. Many of you may be uncomfortable with this lack of control. There are two points to be made in favor of this setup. First, you are already busy enough managing the database, so the burden of controlling physical storage is something you should welcome passing off to it. Second, you can always control which disk receives the data by creating tablespaces that have only one data file (if you really have to be certain on which disk a table is located).

It will be useful now to go over a command that enables you to see what tablespaces you have on your system. A screen with the command and sample output from my Personal Oracle7 database are shown in Figure 10.2. This illustrates an important general concept when dealing with Oracle. To find out what you want to know, you usually issue a Structured Query Language (SQL) query. Some prefer to use Oracle's SQL*Plus product and others prefer SQL*DBA or Oracle Server Manager. However, either way you are accessing a view created for the database administrator that searches the internal Oracle tables to find the information that you need. I keep a list at my desk of the DBA views for those times when I am going after a view I rarely use. These are your windows into what is happening with your database.

Figure 10.2.
Listing the tablespaces
in an Oracle database.



The screenshot shows an Oracle SQL*Plus window with the following content:

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from sys.dba_tablespaces;

```

TABLESPACE_NAME	INITIAL_EXTENT	NEXT_EXTENT	NIN_EXTENTS	MAX_EXT

STATUS				
SYSTEM	4096	4096	1	
ONLINE				
USER_DATA	4096	4096	1	
ONLINE				
ROLLBACK_DATA	102400	102400	2	
ONLINE				
TEMPORARY_DATA	4096	4096	1	
ONLINE				
SQL>				

One of the first things you will notice about the listing in Figure 10.2 is the series of columns that deal with extents or `pct_increase`. These are the default *storage parameters* for the tablespace. Every object that takes up space in the database (for example, tables and indexes) uses preallocated chunks of space known as *extents*. Oracle could have been designed to put rows of data into a big tablespace in the order in which they were received. However, with larger systems, you would have to scan a large amount of disk space to find all the rows associated with a particular table. Recall that Oracle chose to group the rows of a table into areas on the disks known as extents. This helps because Oracle reads extra rows into the SGA buffer cache (which makes sense because most operating systems read entire blocks of data from disk at a time).

Oracle could have stopped with the idea of having everything in a single extent for a database object; however, that would force developers to size each table accurately at the beginning of the project. You also would have to preallocate a large amount of space at creation for tables that may grow slowly over time. The compromise position that Oracle adopted was to make extents even multiples of the operating system block size and enable each table to occupy a reasonable number of extents. The maximum number of extents that a given object can occupy varies by operating system, but can usually go to over 100 extents.

Now that you have the concepts of storage under your belt, you are ready to understand the default storage parameters that are shown in Figure 10.2. These are only defaults; you can override them for database objects that you create (as is shown in the sections that follow). If you do not specify the default storage parameters when

you create the tablespace, you will get the default values associated with the database. If you do not specify default parameters for your database, you inherit a set of defaults that Oracle provides that are tailored to your operating system. The following are the default storage parameters for a tablespace:

- ◆ `initial_extent` enables you to specify the size of this first extent. You have to allocate at least one extent when you create database objects. Note that you cannot create an extent larger than the largest contiguous extent that is available in a single data file. You may have a lot of space available, but if it is scattered between disk files and is interrupted with other extents, you will be limited to a small extent size for new objects.
- ◆ `next_extent` specifies the size of the second extent that you allocate. All further extents are sized according to the value for this extent increased successively by the percent increase factor discussed later.
- ◆ `min_extents` is the minimum number of extents you specify. You do not have to create just one extent when you create an object. I almost always create one extent, but you can specify any number of extents in this field, limited only by the `max_extents` parameter.
- ◆ `max_extents` is the maximum number of extents that this object can occupy. As stated in the earlier discussions, if you have a large number of extents, it may take some time for the disks to locate all the extents that you may need in a query and send back the data. Generally speaking, the fewer extents that there are for an object, the more rapid the response time will be.
- ◆ `pct_increase` is a parameter that I always set to zero, but some people use it. This factor represents a percentage increase to apply to each successive extent allocated. For example, consider a table where the `next_extent` size is 1000 bytes and the `pct_increase` is 10. Your second extent would be 1,000 bytes, your third extent would be 1,100 bytes, your fourth extent would be 1,210 bytes and so on. I do not like to use it because I like to keep the number of extents down to a reasonable value for performance and know exactly what the size of the next extent to be allocated is without having to go to a calculator. For example, the 20th extent allocated in this example would consume over 5,500 bytes. Imagine what the 500th extent would be.

Another feature to notice in Figure 10.2 is the status column. You do not have to have all tablespaces online at the same time. If you do not want data altered for a given application that has its own tablespace, you can take the whole tablespace offline. The users will be told that they cannot gain access to any objects in that tablespace while it is offline. This comes into play later when you learn about recovery. If a single data file is damaged, you can bring the rest of your Oracle instance up and recover that single tablespace.

Having completed the discussion of tablespaces, you may still be wondering where you put the rows and columns of data that you want to store. The answer to this question is Oracle tables. Tables are the fundamental resting place of user data within Oracle. The other data storage objects (indexes, views, and so forth) are derived from or are supporting tools for the table data. There is an entire section coming up next to discuss tables.

Data storage objects contain no data of their own. Instead, they reference data that is contained in one or more tables. The following are data storage objects:

- ♦ *Indexes*, which are sorted lists of the rows in a table
- ♦ *Views*, which are selected columns of data from one or more logically linked tables
- ♦ *Synonyms*, which are merely alternative names for tables.

Another interesting feature that has come into Oracle beginning with version 7 is the stored procedures, triggers, functions, and packages (collections of multiple-stored procedures that are stored in one logical entity). One of the interested concepts of the object-oriented world is the idea of storing data and the actions that you take on that data together. While Oracle has not yet achieved this true object-oriented function, it has begun the process of getting there by enabling you to store software within the database. A *stored procedure* is one or more PL/SQL (Oracle's programmatic extension to the Structured Query Language) programs that are stored together.

A final set of objects in the database provides useful constructs in certain programming situations. The first of these is sequences. Many programming applications need to have a series of numbers assigned to the rows that are guaranteed to be unique. Programmers could construct tables that have a column with a number that represents the next available value for these columns that are unique. Then when you want to create a row, you can find this new value, increment the counter, and store it in the base table. A *sequence* is a simpler construct that does that all for you. The other construct that is used is the cluster. A *cluster* is a group of tables that you want to store together because they contain a number of common columns and are often joined together in queries.

How do you access database objects? The answer is simple—SQL. You cannot use operating system commands to edit the data file manually. The basis for interaction with Oracle objects is this single language. You may have worked with tools that enable you to access Oracle data using a nice graphical user interface. These tools are merely providing a convenient front end that you use to specify what you want. The tool then takes your input and constructs the SQL that is needed to interact with Oracle.

The next sections explore each of the data objects in a little more detail. There may be some objects, such as clusters, that you never use, but it is useful to at least understand what they can provide for you so that you can decide whether you need them as new situations arise.

TABLES

Whatever the front-end tool used to enter data into the Oracle system, when a user enters Greene for a last name and Joe for a first name and commits the transaction, a table will be the storage place for that data.

You may be interested in knowing how Oracle chooses which bytes on the disk to use to store the file, how it delimits the various columns, and other such detailed storage data, but it's not necessary. When you insert a row into a table, Oracle finds the correct spot within the extents that you have allocated for that table and stores the data in a manner that you can retrieve it. Oracle keeps track of the exact location of the bits and bytes. The only internal storage feature that you might use is the row ID. This unique number specifies a given row in a given table. That way, if you have multiple rows in a table that have the exact same values and you want to get rid of one of them, you can use the row ID to do it.

Here is a sample command used to create a table called `golf_scores`:

```
create table golf_scores
(course          char(20),
date_played    date,
partner        char(20),
front_nine     number(3),
back_nine      number(3),
total          number(3))
initrans 1
maxtrans 100
tablespace user_data
storage (
initial 10K
next 10K
pctincrease 0
minextents 1
maxextents 100);
```

The most straight-forward approach to analyzing this command (and tables in general) is to break it down into its components:

- ◆ `create table golf_scores` tells Oracle what to do (create table). Every table needs a name, which is any valid combination of characters and underlines (spaces are not allowed) that is less than 30 characters. I like table names that tell me what is in there versus the old mainframe code and number

systems. By the way, Oracle has certain reserved words, such as `order`, which you cannot use as the table name (although `order` could be part of a larger name).

- ◆ The next five lines (contained within parentheses) list the names of the columns for this table. Associated with each column is a data type (for example, `char` for character) and size. (The various types of data and sizing are discussed soon.)
- ◆ `initrans` and `maxtrans` are similar to the `initial` and `next` extent parameters, except they allocate space to record transactions that are in progress to the table. This space is allocated in the header of the table and is in chunks of 23 bytes each. This space is not recovered when the transaction is completed. The maximum value for these is no more than 255 and may be less on some operating systems. If you run out of transaction space for a table, new transactions have to wait until transaction space is freed up by existing transactions completing. If you do not specify values for these parameters, default values that apply to your instance are used.
- ◆ `tablespace user_data` is where you tell Oracle to put your table—in this example, a tablespace called `user_data` (this is the Personal Oracle7 equivalent of users in most multi-user Oracle systems). If you do not specify this parameter, the default tablespace for your user ID will be used (which is discussed in Chapter 26).
- ◆ `storage()` contains five parameters within the parentheses that are the storage parameters for this table. If you do not specify any of these values, the default value for that parameter in this tablespace is used.
- ◆ `initial` and `next` are the initial extent and next extent parameters described under tablespaces.
- ◆ `pctincrease` is the percentage of increase in size applied to each of the next extents.
- ◆ `minextents` and `maxextents` are the minimum and maximum number of extents that you have for this table.

Each column in a table has a column name, a data type, and some sizing parameters. The rules for column names are the same as the rules for table names and it's a good idea to use column names that are meaningful. Character data has a size to it that specifies the maximum length of the character string. Numbers have precision and scale. *Precision* refers to the total number of digits, not including the exponent when this number is expressed in exponential notation. The *scale* specifies the number of digits to the right of the decimal place. A number with a precision of 5 and a scale of 2 will range between -999.99 and +999.99 and be considered to be data type `number(5,2)`. Table 10.1 lists data types supported by Oracle (older versions may not support the `long raw` or `varchar2` data types):

TABLE 10.1. ORACLE DATA TYPES.

<i>Data Type</i>	<i>Meaning</i>	<i>Fixed / Variable Length</i>
<code>char(size)</code>	Character	Fixed length (trailing spaces added)
<code>varchar2(size)</code>	Character	Variable length (uses what is needed)
<code>number(prec,scale)</code>	Number	Variable length (uses what is needed)
<code>date</code>	Date and time	Seven bytes per column
<code>long</code>	Character	Variable length up to 64K in Oracle 6 or 2G per row in Oracle 7
<code>raw(size)</code>	Free-form	Variable length up to 2,000 bytes per row
<code>long raw</code>	Free-form	Variable length up to 64K in Oracle 6 or 2G per row in Oracle 7
<code>rowid</code>	Binary	Six bytes per row that can be represented as a number in queries

Access to the tables in most cases is with the Structured Query Language (SQL). This language has a number of statements that you need to learn to work with Oracle tables properly (which are included in Appendix A). It is important to understand that when you issue the `select` command to retrieve a row from a table, you actually get one or more blocks of rows back from the data files. The number of blocks that you retrieve is set up by some of those database tuning parameters that are discussed in Chapter 32.

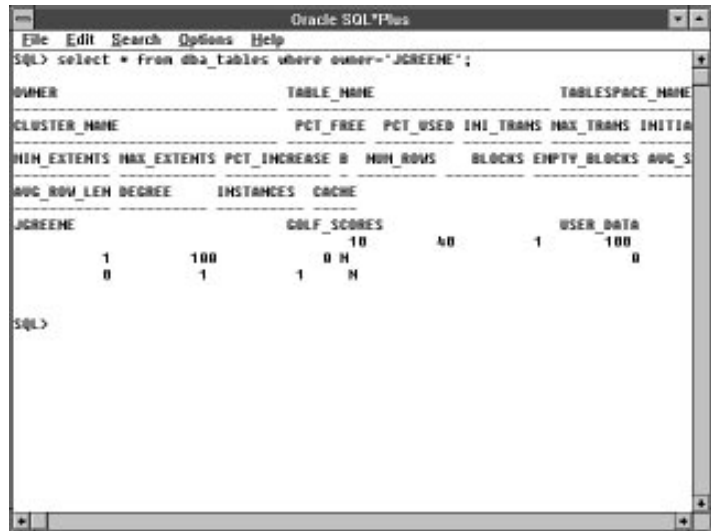
Of course, you do not want every user in your instance creating tables all the time. Therefore, Oracle enables the DBA to control who can create or modify these objects via the privileges mechanism. Chapter 11 discusses privileges in some detail, but for now, understand that there are two basic privileges related to table creation and modification:

- ◆ CREATE TABLE
- ◆ ALTER TABLE

One final topic before we leave this section. It is useful to be able to determine what tables are out there and how they are set up. Once again, there is a DBA view that provides the key information that the DBA needs to manage tables. Figure 10.3 shows a screen with a sample of a query to that view to verify that the table created

earlier in this section is set up the way you intended. Note that the `where` clause is used to narrow the range of information that you want to see. The format of the `where` clause is relatively simple: you specify a field name, a comparison, and then a value or list of values. Your SQL Language reference book can give you the complete details on forming more complex Oracle comparisons.

Figure 10.3.
A query against the
`DBA_TABLES` view.



The screenshot shows the Oracle SQL*Plus interface with a query executed: `SQL> select * from dba_tables where owner='JGREENE';`. The results are displayed in a table format with the following columns: OWNER, TABLE_NAME, TABLESPACE_NAME, CLUSTER_NAME, PCT_FREE, PCT_USED, INIT_TRANS, MAX_TRANS, INITIAL_EXTENTS, MAX_EXTENTS, PCT_INCREASE, MIN_ROWS, BLOCKS, EMPTY_BLOCKS, AVG_SPACE, AVG_ROW_LEN, DEGREE, INSTANCES, and CACHE. The query returns three rows of data for the owner 'JGREENE'.

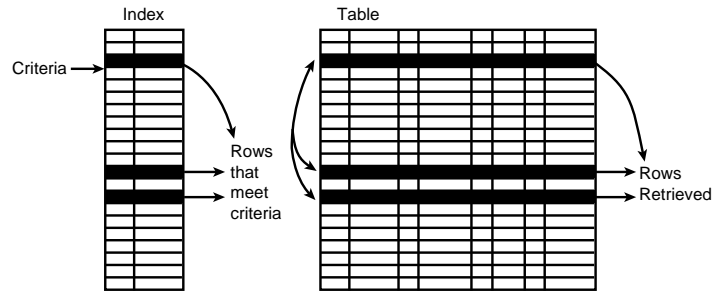
OWNER	TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	PCT_FREE	PCT_USED	INIT_TRANS	MAX_TRANS	INITIAL_EXTENTS	MAX_EXTENTS	PCT_INCREASE	MIN_ROWS	BLOCKS	EMPTY_BLOCKS	AVG_SPACE	AVG_ROW_LEN	DEGREE	INSTANCES	CACHE
JGREENE	GOLF_SCORES			10	40	1	100	0	1	0	1	0	0	0	0	1	1	N
JGREENE	USER_DATA			100	0	1	100	0	1	0	1	0	0	0	0	1	1	N

INDEXES

The `where` clause is actually quite a popular construct in database programming. Very few users want to see all the data in a table displayed for them on every query. Instead, most queries are designed to return a series of rows or perhaps even a single row that matches the users' criteria. For smaller tables, it is not a big deal to read in all the rows and check them off one by one to see whether they match the selection criteria input by the users. However, for large tables (I saw one table once that was 17G), this is extremely slow and places a large load on the system for every query. To help solve this problem, indexes were created.

An index is a sorted list of data from one or more columns in the table that are commonly used as selection criteria. Thus, rather than searching through a large number of long personnel records to find the record of an individual, you may sort through a relatively short index containing just the last name and first name (see Figure 10.4). When Oracle finds the name that matches your search values, it retrieves just the block of rows that contains the data you need. Because the data is already presorted, Oracle does not have to read every record in the index. Instead, it uses a more efficient search routine to reduce the number of index records that it needs to analyze.

Figure 10.4.
Indexed searches.



Because indexes are extracted lists of values for tables, Oracle needs a place to store them. Therefore, indexes have storage clauses similar to tables. Here is a sample command that creates an index on the `date_played` field of the `golf_scores` table of the last section:

```
create index golf_scores_dates
on golf_scores (date_played)
tablespace user_data
storage
(initial 2K
next 2K
minextents 1
maxextents 100
pctincrease 0);
```

How do you get your queries to use these wonderful indexes that you have created? The answer is that Oracle automatically uses indexes if you issue a query that can use the index (that is, the values in your `where` clause are among the fields of the index). That makes life a whole lot easier for developers. Even better, indexes are maintained automatically by Oracle. When you add, delete, or modify rows in a table that has one or more indexes, the indexes are automatically updated to reflect the changes. This brings up an important design consideration. It takes Oracle time to update an index. If you have a table that has a large number of indexes, it takes longer to add data to this table. Therefore, tables wherein the key to performance is the capability of adding data (for example, an order entry on a system that is heavily loaded during the day) benefit from fewer indexes. Conversely, systems that are used primarily for queries (such as decision support systems) benefit from having as many indexes as are useful based on the types of user queries.

It is sometimes difficult to answer the question as to what indexes are useful in a table. Obviously, indexing every column in the table is probably not be a good idea. Even though an index search is more efficient than a table scan, it does take some time. When you add the time of searching a large index to the time it takes to retrieve the rows you need from the table, you usually wind up taking longer than if you just scan the table. It also may not be justifiable to build and maintain indexes that support the selection criteria of infrequently used queries. In this case, it costs more

time on the updates than it saves on these infrequent queries. Being trained as a scientist with a love for experimentation, I always try constructing an index that I am curious about and then see its effect on the performance of my queries and update routines. Based on this data, I decide on whether it was worth keeping.

The following are a few more notes about indexes:

- ♦ Tables are not affected by the presence or lack of indexes. You can drop and re-create indexes at will without affecting any of the data in a table.
- ♦ It is sometimes beneficial in systems where you perform large batch updates to drop the indexes, perform the updates, and then re-create the indexes.
- ♦ One rule of performance tuning is to locate the indexes for a table on a separate disk from the table itself. This splits the input/output load for queries that access both the table and an index.
- ♦ You have the option of creating unique indexes. If you create a unique index, it gives an error if you try to insert a row in the table that has values in the index columns that match those of another row in the table. This is a good way to enforce a primary key on the table and build an index in a single step.
- ♦ If you have privileges to create or modify a table, you have the privileges you need to create or modify indexes associated with that table.

Indexes are very useful tools to improve the performance of queries. However, like all tools, you need to consider their design carefully to ensure that you get the performance gains that you desire. Figure 10.5 shows a screen with the query that you would issue to view the indexes associated with a particular user in the database.

VIEWS

Next on your list of database objects to study is views. You have already seen a number of views in the examples where we queried the database for a list of tables, tablespaces, and indexes. The actual data retrieved is stored in one or more internally controlled Oracle tables, and I would have to do some research to figure out what they are. I honestly don't care. I have these nice little views that I issue simple queries against to get my answers.

A view is merely a selection of one or more rows from one or more tables. The data is not duplicated. The view is merely a structure that points to the data that is

actually stored in the tables. Therefore they do not require storage parameters and do not reside in tablespaces. Views can be used to achieve a number of objectives for designers:

- ◆ Your users do not wish to write complex queries that join two or more tables together. Therefore, you build the logic joining the two tables into the view and let the users access the data as if it were a single table.
- ◆ You may have tables where certain users are permitted to access data only from certain columns. You can create views that contain the columns that they are allowed to access and give those users access to only the view and not the base table.
- ◆ You have tables where the column names may not make sense to certain users. You can create a view that renames these columns to names that make sense.
- ◆ You may wish to deal with simpler structures than your tables. For example, your table may have hundreds of columns of data. It may be easier to create a view and then have the users worry about only the simpler view.

Figure 10.5.
Determining the
indexes owned by a
user.

OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	TABLESPACE_NAME	MAX_TRANS	INITIAL_EXTENT	NEXT_EXTENT	NIN_EXTENTS	MAX_EXTENTS	PCT_INCREASE	PC
JGREENE	GOLF_SCORES	JGREENE	GOLF_SCORES	TABLE	NONUNIQUE	USER_DATA	255	4096	2048	1	100	0	
STATUS													
VALID													
SQL>													

One of the best features about views is that you access them the same way you access tables. You do not have to learn any separate language constructs. You cannot create indexes on views. To create a view, you need the CREATE VIEW privilege assigned

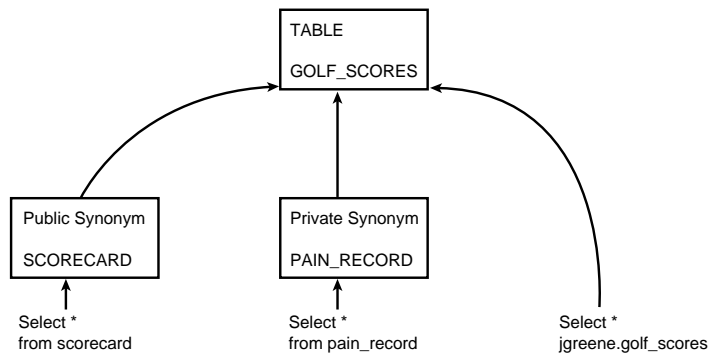
to your Oracle user ID. You cannot alter a view. Instead you just drop it and re-create it. You can do this in a single command by using the `CREATE OR REPLACE VIEW` format of the `CREATE VIEW` command. The advantage of the `CREATE OR REPLACE VIEW` format is that you do not lose the privilege grants that may have been granted on the view. This is not a problem, because dropping or creating views has no effect on the data in the tables associated with that view. By the way, DBAs have all these privileges and more, as is explained in Chapter 11.

Views are convenient database objects that do not store any data themselves. Instead, views point to data located in one or more tables in the instance. They have many uses, including making queries more convenient for the users and providing additional security for the database.

SYNONYMS

A synonym is merely a way of assigning an alias with which users can reference a table or view. This makes it easier to access tables that you do not own. When you own a table, you use its name in your SQL queries. When you want to access a table owned by another user, you need to specify both the owner of the table and the table name (in the `owner.table_name` format). When you reference tables located in a different database using SQL*Net, the format can be rather complicated. Synonyms simplify this access because you can create a simple synonym that contains the details of a complex reference string (see Figure 10.6).

Figure 10.6.
Synonyms.



There are two types of synonyms—private and public. *Private synonyms* are those that you create for your own personal use. *Public synonyms* are those that you create that any user in the database can utilize. Obviously, from a DBA point of view, you usually want to restrict the number of users who can create public synonyms. You usually want to be even more restrictive with the number of users who can drop public synonyms. There are three primary system privilege sets that the DBA is concerned with related to synonyms:

- ◆ CREATE SYNONYM (which also allows you to drop private synonyms)
- ◆ CREATE PUBLIC SYNONYM
- ◆ DROP PUBLIC SYNONYM

Finally, it is interesting to consider what you get when you issue a query that goes against an object name that states the following: you own a table with that name, you have a private synonym with that name, and there is also a public synonym with that name. I had to try this out once to figure out what Oracle would do. After I got the results of this experiment, it made sense to me. So, here is the order in which Oracle accesses objects that have the same name in the database:

- ◆ If you own a table with a particular name, you always access your table on a query to that object name.
- ◆ If you do not own a table with that name, but have a private synonym with that name, you access the object associated with that private synonym.
- ◆ Finally, if you have neither a table nor a private synonym with the object name you are searching for, you access the object specified by the public synonym.

STORED PROCEDURES

As mentioned earlier in this chapter, stored procedures are a way of storing software within the Oracle database. This feature applies only to Oracle version 7.0 and later databases. Basically, anything that you can develop in a PL/SQL script can be saved as an Oracle stored procedure. A collection of stored procedures can be saved as a package in the database. Stored procedures can present several interesting advantages for Oracle developers:

- ◆ Oracle security mechanisms, such as roles and grants, can be used to protect both the data and the software. This also provides one place of control for security administration.
- ◆ When you execute a stored procedure, you execute it as if you were the person who created it and not with the privileges that you would normally have. This fits well with certain environments, which need to implement security features in the software, such as audit trails and special validity checking. The DBA can grant write access only to the user who creates the stored procedures. Users who are authorized to perform write operations are not given that permission in their Oracle accounts. Instead, they are given access to the stored procedures that must be executed (and hence the software security features enforced) to perform the write operations. If they try issuing SQL commands to write to these tables from SQL*Plus, they are denied access.

- ◆ Another advantage of stored procedures is that Oracle stores both the source code and a parsed version of the software. This saves Oracle the time of having to parse the statements each time as it would for SQL queries received from external software applications.

There are two problems with stored procedures. First, all stored procedures are stored in the system tablespace, usually thought of as sacred ground for use only by the DBA. However, the Oracle developers do not agree. Therefore, you have to let non-DBA software be placed in your most special tablespace. The other problem is that the software has to be written in PL/SQL. PL/SQL gets the job done; however, it is difficult to debug and does not support normal input/output operations. Stored procedures are very useful if you are willing to live with these two annoyances.

Stored procedures require special system privileges for creation (as you would expect with something that can be this powerful and that has to be placed in the system tablespace). Without getting into the details that are of concern to software developers, you need to grant `CREATE PACKAGE`, `CREATE PROCEDURE`, `DROP PACKAGE`, and/or `DROP PROCEDURE` to users who will be developing stored procedures in your system.

CLUSTERS

Recall that a *cluster* is a group of tables that you want to store together because they contain a number of common columns and are often joined together in queries. I have never used clusters. I never had the need or opportunity when working with Oracle 6 and I have been told that they are not as important with Oracle 7. The reason for this is that the basic data storage algorithm for Oracle 7 is a lot smarter than in version 6 and it will arrange tables in an efficient manner without creating clusters. One nice feature about clusters is that users have no idea whether the tables that they are accessing are in a cluster or not. Only the developers and DBAs know which tables are in clusters.

The basic Oracle references provide a good description of the details of setting up clusters. They also go into more detail about when you might use them. However, for purposes of this survival guide, it is enough for you to know what they are. If you have performance problems with queries that always join together a few tables that have common columns, you may consider experimenting with clusters. Otherwise, you may never see a cluster. (Although many of the Oracle internal tables are supposed to use clusters, you may never deal with these tables directly because that is what you purchased the RDBMS software to do for you.)

SEQUENCES

The final database object to consider is the sequence. As discussed in the introduction, sequences are tools used to generate a unique sequential number that you can use in your data tables. It is an excellent way to generate a guaranteed unique number to use as an internal key to link two tables in an application. One of the best features of sequences is that they can guarantee that you will get a unique value when you access the sequence. If, for example, you create your own sequence table and write applications that read the next available value, increment it by one, and then update the sequence table with this new value, there is always a chance that someone else can access the same record that you did before your update statement is issued (you could get around this if you played fancy games with locks, but why bother when sequences are available). If this is a primary key to a table, you will have a corrupted primary key. One final note: you cannot roll back your incrementing of a sequence value. If you decide not to insert the row with that sequence value into your table, you do not recover the sequence number.

Sequences are easy to access. For example, if you create a sequence called `staff_id` to track a unique, internal key for each staff member entered into a personnel database, you can enter a new record of data and generate a unique key with a simple SQL statement such as the following:

```
insert into personnel
values(staff_id.nextval,
'Jones',
'Sandra',
'123-4567');
```

As a final note on sequences, there are a series of Oracle system privileges that are needed to create or work with sequences. These privileges are `CREATE SEQUENCE` and `DROP SEQUENCE`. This should be enough to get you started and enable you to understand what sequences are. There is additional background material in the basic Oracle documentation set if you want to try to get fancy with sequences.

SUMMARY

This chapter provided you with an overview of how Oracle stores data logically and also presented the types of objects that are available for storing and retrieving data in Oracle. These database objects are grouped into four categories:

- ♦ Tables, which serve as the primary location to store data
- ♦ Derived storage objects, such as indexes and views
- ♦ Software stored within the database in the form of stored procedures and packages
- ♦ Useful programming constructs in clusters and sequences

By now, you should be comfortable with at least the basic objects, such as tablespaces, tables, indexes, and views. If not, review these sections again because most of the discussion in the next several chapters focuses on managing database objects through privileges.

- 
- Overview of Oracle Privileges
 - Overview of System Privileges
 - Typical Privilege Sets
 - The “Any” Privilege Sets
-

CHAPTER 11

Oracle System Privileges

In the days when card decks stored all data, it was easy to control access. You simply locked up the card decks. Since that era, however, computer people have struggled to balance the need to let people have access to the data needed to be productive against the need to protect certain sensitive data items. The Oracle RDBMS provides some pretty sophisticated control mechanisms to enable you to provide just the right amount of access to your database. However, to get that scheme set up requires a fair amount of work by someone. In most installations, there is not a separate security administrator (or at least there is not a security administrator who gets into the actual technical work of granting this access to that person). Therefore, this assignment usually falls on the database administrator.

This chapter is designed to provide an introduction to the first set of security mechanisms that Oracle provides to control access to data. The system privileges are designed to control overall database access permissions. Examples of these are permissions to create and delete tables. The next chapter explores the Oracle object privileges with which you grant a particular user the permission to read data from a particular table on the system. Finally, Chapter 13 ties the concepts of the two types of privilege in with the idea of roles and the actual commands that can be used to set up access for a particular user. Taken together, these three chapters enable you to control access to your Oracle database effectively.

This chapter begins with an introduction to some of the concepts and purposes behind Oracle access privileges. Then you review the first type of Oracle access privileges—those related to the system as a whole. An entire section is devoted to the powerful (and therefore dangerous) group of system privileges that I refer to as the “any” privileges. Finally, several examples of combinations of privileges that I have run across in my travels are presented. This will be a starting point for you when you start working out the privileges that you will give to your various users.

Before we jump right into system privileges, let’s go over the reason that Oracle has implemented this set of privileges. The quick and simple answer is that customers wanted it and Oracle wanted to sell its database product. There was a balancing act involved with making it simple to get small, nonsecure organizations working quickly and still provide detailed control for those that need it. Finally, it is important to understand that the access scheme has been significantly upgraded between versions 6 and 7 of the Oracle RDBMS. Therefore, the sections that follow cover each of these topics separately.

OVERVIEW OF ORACLE PRIVILEGES

If everyone did exactly what they were told, never made any mistakes, and could be absolutely trusted, there would not be a need for security systems in database

management systems. Now back to the real world. Certain people lack the knowledge or need to create and drop database objects such as tables and indexes. Additionally, most business organizations limit who can read or update certain types of information based on confidentiality laws, financial responsibility considerations, or management preferences.

In the era where computers were not accessible by mere mortals, the locked data center combined with controlled batch jobs for update and report output provided adequate security. Then information systems were made available throughout the organization to increase productivity. Although the vast majority of users violated no access rules or performed no accidental deletions, there were a few painful problems that received a lot of management attention. This caused businesses to tighten their security picture.

This was difficult in the flat-file era because the operating systems at first either allowed access or didn't. Eventually, the operating systems became smart enough to distinguish between read and write privileges, so this at least controlled those who could not perform updates. When developers received a requirement to limit access to a certain portion of the data, they could always move that data to a new file and limit the operating system write access to that file.

The trend toward database management systems created a need for the database to control access to various tables internally. This came from the fact that businesses usually did not want to have a separate operating system file for each table in the database. Therefore, control mechanisms were created to limit various types of access to the tables. In addition, because the majority of database users had no need to create database objects, certain restrictions were developed to prevent such activities.

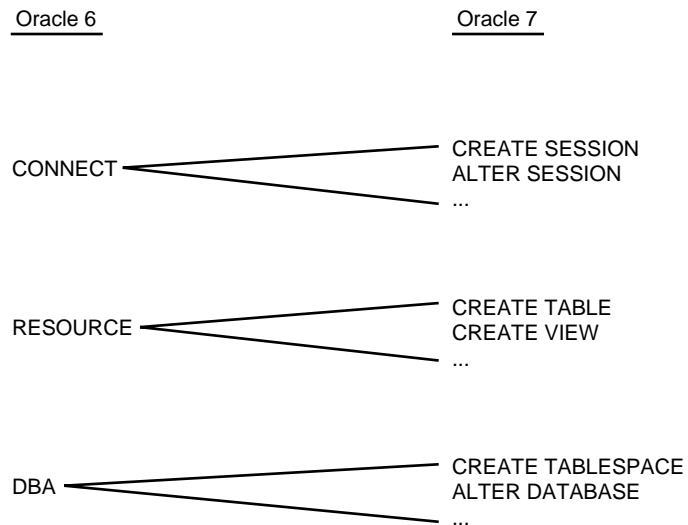
This brings us to the era of Oracle 6. In this world, there are three classes of users. The *connect users* are allowed to access the system (read and write as controlled by grants to the individual objects), but they cannot create any new database objects. The *resource users* are allowed to read and write data as controlled by individual object grants, and they can also create new tables, indexes, and so forth. Finally, the DBAs have the keys to the kingdom and can access any of the objects on the system, create new objects, and also perform all the administrative functions that are needed to keep the database operational.

This was a functional system; however, it had some severe limitations for larger databases and information systems. One of the classic large-system problems was that the operators needed to have DBA privileges to perform functions such as database startup and shutdown. This was uncomfortable because even though these were trusted individuals, they usually lacked the training to have all the

power of the DBA account. With only three categories, you also had no way of qualifying various types of developers or limiting their capability of creating certain types of database objects.

The Oracle 7 privilege scheme was enhanced to enable a much greater degree of control over the privileges assigned to individuals. It broke up the three types of system privilege in Oracle 6 (connect, resource, and DBA) into 78 specific privileges. If, for example, your organization allows operators to perform database startups, shutdowns, and backups, you have the fine control to grant just these privileges to the operators. Figure 11.1 illustrates this concept. Operators do not have the DBA's privilege of deleting any data table in the system, for example.

Figure 11.1.
Comparison of Oracle 6
and 7 system privileges.

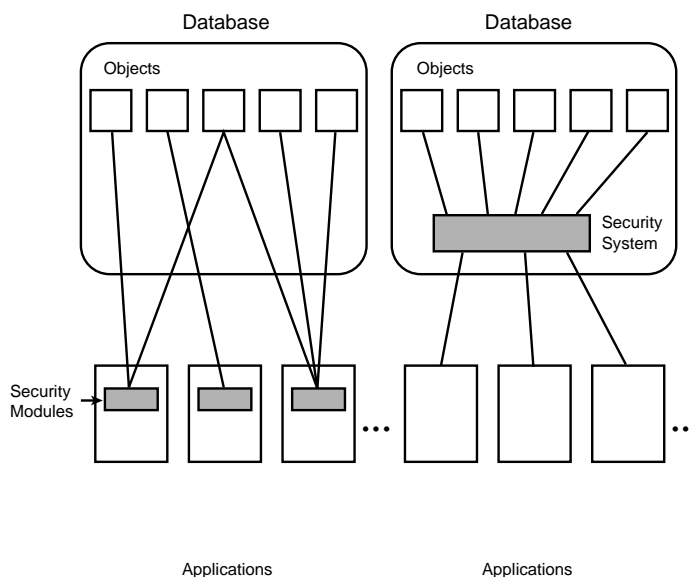


With this fine level of control came the burden of administering all these privileges over a large number of users. That is where the Oracle 7 concept of roles comes into play. The example given in the previous section showed the set of privileges required by computer operators. In most organizations, the DBA grants privileges based upon an individual's function within that organization (computer operator, developer on the sales data warehouse project, manager of general ledger accounting, and so forth). Therefore, it is useful to be able to say that all people who have a certain job function have a common set of privileges. You assign a certain set of privileges to the role and then assign individuals to that role. More on roles in lucky Chapter 13.

The next concept that ties in with system privileges is where the security is going to be implemented. In many traditional mainframe systems, the security is built into the applications. This enabled each organization to implement very specific data access rules for each application. However, this scheme required a large amount of testing and maintenance of the security mechanisms that were put into place. Oracle's concept has been that you buy most of the security mechanisms as part of the RDBMS product. You may have to do some administration, as is the case when creating roles and granting access to the users, but this is much less complex than writing and testing the basic security software. Only in a few specific cases that I have run across (mostly in areas that involve money, such as banks) is there a need to add locally developed software to enhance the basic security mechanisms.

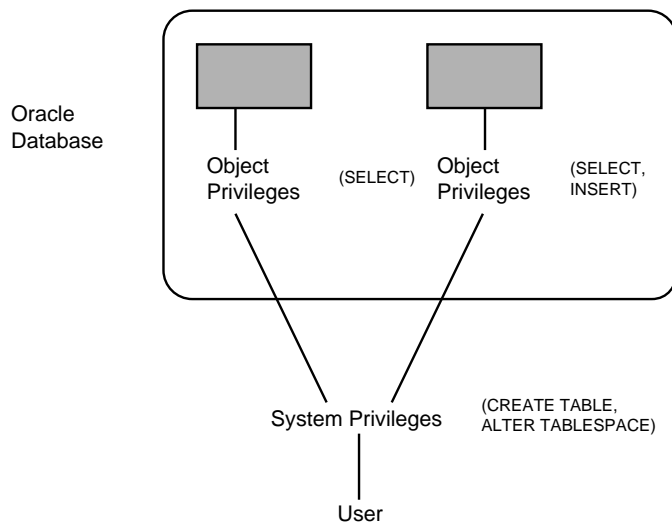
This becomes especially important as greater access is provided to the database, especially through client-server mechanisms. It is the classic tradeoff. You need to protect your data, but the data is useless if people cannot access it. The moment you provide general-purpose access tools to your users, such as SQL*Plus or Excel with a link to the Oracle database, you have given the users the means to bypass your application security (see Figure 11.2). The users can connect to the database with whatever privileges you assign them. They can then access or update data using tools that are designed to allow them to do whatever they want. Therefore, it is important to develop a sound security scheme for databases that provide this type of access. Some solutions to this problem will be found in Chapter 13 after the discussion of the security mechanisms is complete.

Figure 11.2.
Database versus
application security.



It is important to understand that the *access privileges* comprise only half of the Oracle database security picture. These privileges control what you can do to the database as a whole. The other set of privileges, the *object privileges*, provide a fine level of control as to what a user can do with each individual database object (tables, views, and so forth). Not all objects are directly controllable. Indexes, for example, are used automatically whenever users have permission to access their associated tables. Figure 11.3 illustrates the relationship between these two types of Oracle privileges. (Chapter 12 discusses object privileges further.) The rest of this chapter is devoted to presenting the Oracle system privileges from the database administrator's point of view.

Figure 11.3.
Oracle system and
object privileges.



OVERVIEW OF SYSTEM PRIVILEGES

As discussed in the last section, Oracle system privileges control a user's access to the Oracle instance as a whole. They specify the Oracle functions (for example, `CREATE TABLE`) that a user can perform. Before going into the various system privileges, note these points:

- ◆ Certain privileges have prerequisites of other privileges. For example, you must have `CREATE TABLE` to have the `CREATE SNAPSHOT` privilege.
- ◆ Read the descriptions of the privileges. There are a few privileges (such as `GRANT ANY PRIVILEGE`) that are not exactly what you would expect by just reading their titles.
- ◆ The key to being able to grant an Oracle system privilege in Oracle 6 is having the `DBA` privilege.

- ◆ The key to being able to grant an Oracle system privilege in Oracle 7 is having the GRANT ANY PRIVILEGE privilege assigned to your account.
- ◆ There are additional mechanisms to provide finer levels of control that are available to the DBA. For example, if you want to allow users to create tables, but only in a certain tablespace, you can give them the CREATE TABLE privilege and then give that a space quota only on the desired tablespace.

First, let's review the privilege sets that are available under Oracle 6. Oracle 6 users fall into three categories:

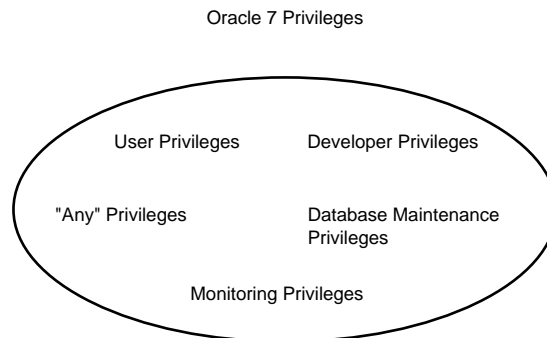
- ◆ **CONNECT:** these users can access Oracle data as permitted by the object privileges that are assigned to them. They cannot create database objects or perform any administrative functions.
- ◆ **RESOURCE:** these users can access Oracle data as permitted by their object privileges and also create database objects such as tables and views. They cannot perform any database administrative functions.
- ◆ **DBA:** these users can access data, create objects, and perform Oracle administrative functions. They have full access to all data in the database and can delete any database objects that they want. There are very few restrictions placed on users with the DBA privilege, so be extremely careful when using these accounts.

Now on to Oracle 7 and its more sophisticated privilege scheme. Rather than present system privileges as a laundry list, one after the other, these privileges are grouped into functional categories (see Figure 11.4).

- ◆ *The User Privileges.* These privileges relate to users who will access the system, but not create any objects. These are similar to the CONNECT users under Oracle 6.
- ◆ *The Developer Privileges.* These privileges relate to the creation of database objects. These are similar to the RESOURCE users under Oracle 6.
- ◆ *The "Any" Privileges.* These are an especially powerful set of privileges within Oracle. Normally, I would give these only to DBAs and extremely senior developers working in dedicated development instances. These are similar to regular privileges except that they apply not only to objects that you own, but objects that other users (including SYS and SYSTEM) own.
- ◆ *The Database Maintenance Privileges.* These privileges relate to the care and feeding of the database itself. They are typically assigned only to DBAs and other computer support staff.
- ◆ *The Monitoring Privileges.* These privileges relate to keeping an eye on the Oracle database. They are typically assigned to DBAs, but can also be used by separate security administrators to monitor system security.

The next sections explore each category individually.

Figure 11.4.
System privilege
categories.



THE USER PRIVILEGES

The first set of privileges belongs to common users of the database (see Table 11.1). Even though they share this common and relatively simple function (read and write to existing objects), Oracle provides a few variations in access privileges that allow organizations to tighten up security to the maximum extent possible (for example, by removing the ALTER SESSION privilege).

TABLE 11.1. THE USER PRIVILEGES.

<i>Privilege</i>	<i>Description</i>
CREATE SESSION	Allows the user to connect to the Oracle database. Without it, a user has no access to Oracle.
ALTER SESSION	Allows the user to issue the alter session command, which permits such actions as setting National Language Support parameters, controlling database links, and using the trace facilities.
FORCE TRANSACTION	Allows the user to commit or roll back a distributed database transaction in the local database. Not a commonly used feature.

THE DEVELOPER PRIVILEGES

The user privileges are rather simple and straightforward. The next set of privileges relates to developers who may be working with the system (see Table 11.2). These can be a little trickier in that you may not wish to give out all the developer privileges in your instance (there are some rather powerful ones).

TABLE 11.2. THE DEVELOPER PRIVILEGES.

<i>Privilege</i>	<i>Description</i>
CREATE CLUSTER	Create clusters of tables that the developer owns. The developers can also drop clusters that they own.
CREATE PROCEDURE	Create stored procedures, packages, and functions owned by the developer. The developers can also drop any of these objects that they own.
CREATE DATABASE LINK	Define a database link. This feature is similar to a synonym because it is a named pointer to the other database. A key difference is that you store the Oracle ID and password for the remote system as part of the link, thereby establishing access privileges in the remote system. The developers can also drop any database links that they create.
CREATE PUBLIC SYNONYM	Create an alternative name for referencing a database object such as a table or view. Any user in the instance can use this name to make a call for that object. Object privileges are still needed for that user to have access to an object. In certain situations, you may wish to give this privilege only to DBAs or certain senior developers.
DROP PUBLIC SYNONYM	Delete an alternative name for referencing a database object that is available to all users in that instance. In certain situations, you may wish to give this privilege only to DBAs or certain senior developers.
CREATE SEQUENCE	Create a sequence owned by the developer. The developers can also drop any sequences that they create.

continues

TABLE 11.2. CONTINUED

<i>Privilege</i>	<i>Description</i>
CREATE SNAPSHOT	Create a local copy of a table located on another Oracle instance. The developers can also drop any snapshots that they own.
CREATE SYNONYM	Create a private synonym (for the developer's use only). The developers can also drop any private synonyms that they own.
CREATE TABLE	Create a table. The developers can also create indexes on and drop tables that they own.
CREATE TRIGGER	Create a trigger (command executed when a specified event occurs) owned by the developer. The developers can also drop triggers that they own.
CREATE VIEW	Create a view owned by the developer. The developers can also drop views that they own.
UNLIMITED TABLESPACE	Allows the developers to create objects that consume as much space in any tablespace as is available. This overrides any quotas assigned in that tablespace. You could consider this to be very much like an "any" privilege.

These are the privileges that are normally considered for developers. You may find cause to grant some of the "any" privileges described in the next section or even certain database maintenance privileges to certain developers. However, this is the common list that you should consider and a good place to start.

THE "ANY" PRIVILEGES

One of the candidates for the most powerful but dangerous features in the Oracle system is what I call the "any" privileges (see Table 11.3). If you grant `CREATE TABLE` to developers, they have the power to destroy (via the `SQL drop table` command) tables that they have created. However, if they had the `DROP ANY TABLE` privilege, they can destroy any table in the system. This is a serious

privilege. DBAs need to think of this every time they execute destructive commands because, by default, they have most of these “any” privileges assigned to them. In many cases, this is needed. What happens when a developer leaves the company and has a lot of personal tables that are no longer needed? The DBA can delete these tables and drop the user from the system. However, be aware that if you drop a table, especially one referenced by a public synonym, you may not get the table that you anticipated and your DROP ANY TABLE privilege allows you to do it. Enough said. Please be careful when you use your “any” privileges.

Warning

DBA accounts have most of the “any” privileges by default. This enables you to do almost anything in the system. Be extremely careful that your syntax and purpose is correct before issuing a command that takes advantage of your “any” privileges.

TABLE 11.3. THE “ANY” PRIVILEGES.

<i>Privilege</i>	<i>Description</i>
ANALYZE ANY	Allows the user to collect optimizer statistics, validate the structure, or identify migrated and chained rows in any table or cluster in the database.
AUDIT ANY	Allows the user to perform object auditing on any object in the database.
CREATE ANY CLUSTER	Allows the user to create a cluster and assign ownership to any user in the database.
ALTER ANY CLUSTER	Allows the user to alter clusters owned by any user in the database.
DROP ANY CLUSTER	Allows the user to drop clusters owned by any user in the database.
CREATE ANY INDEX	Allows the user to create an index on any table in the database and assign ownership to any user in the database.
ALTER ANY INDEX	Allows the user to alter any index owned by any user in the database.
DROP ANY INDEX	Allows the user to drop any index owned by any user in the database.

continues

TABLE 11.3. CONTINUED

<i>Privilege</i>	<i>Description</i>
GRANT ANY PRIVILEGE	Allows the user to grant any system privilege to any user in the database. This is the prime requirement for DBAs being able to grant system privileges. Note, this does not allow the users to grant object privileges. Only the object owners can grant object privileges.
CREATE ANY PROCEDURE	Allows a user to create a procedure, package, or function and assign ownership to any user in the database. Has the prerequisites of ALTER ANY TABLE, BACKUP ANY TABLE, DROP ANY TABLE, LOCK ANY TABLE, COMMENT ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, or GRANT ANY TABLE, depending on what the function actually does. Note that CREATE ANY PROCEDURE is good enough to create a procedure that accesses database objects (for example, tables) that are open for public access. The prerequisites come into play when you try to access (select or update) a database object and you do not have an explicit access grant for your user ID.
ALTER ANY PROCEDURE	Allows the user to modify any procedure, package, or function owned by any user in the database.
DROP ANY PROCEDURE	Allows the user to drop any procedure, package, or function owned by any user in the database.
EXECUTE ANY PROCEDURE	Allows the user to execute any procedure, package, or function owned by any user in the database. This overrides the object privileges assigned to the procedure, package, or function by the owner.
ALTER ANY ROLE	Allows the user to modify any role created in the database.

<i>Privilege</i>	<i>Description</i>
DROP ANY ROLE	Allows the user to drop any role in the database.
GRANT ANY ROLE	Allows the user to assign a role to another user in the database.
CREATE ANY SEQUENCE	Allows the user to create a sequence and assign ownership to any user in the database.
ALTER ANY SEQUENCE	Allows the user to modify any sequence owned by any user in the database.
DROP ANY SEQUENCE	Allows the user to drop any sequence owned by any user in the database.
SELECT ANY SEQUENCE	Allows the user to select values from any sequence owned by any user in the database.
CREATE ANY SNAPSHOT	Allows the user to create a local copy of a table in another instance and assign ownership to any user in the database. The user must have CREATE ANY TABLE privileges.
ALTER ANY SNAPSHOT	Allows the user to compile any snapshot owned by any user in the database.
DROP ANY SNAPSHOT	Allows the user to drop any snapshot owned by any user in the database.
CREATE ANY SYNONYM	Allows the user to create a private synonym and assign ownership to any user in the database.
DROP ANY SYNONYM	Allows the user to drop any private synonym owned by any user in the database.
CREATE ANY TABLE	Allows the user to create a table and assign ownership to any user in the database. Note that the assigned owner's privileges and quotas will be used to see whether the table can be created.
ALTER ANY TABLE	Allows the user to modify the structure of any table owned by any user in the database.

continues

TABLE 11.3. CONTINUED

<i>Privilege</i>	<i>Description</i>
BACKUP ANY TABLE	Allows the user to use the Export utility to back up tables owned by any other user in the database. This is needed by DBAs or operators who use the Export utility for database backups.
DROP ANY TABLE	Allows the user to drop any table owned by any user in the database.
LOCK ANY TABLE	Allows the user to issue locks to any table (or rows within the table) owned by any user in the database.
COMMENT ANY TABLE	Allows the user to enter comments on any table owned by any user in the database.
SELECT ANY TABLE	Allows the user to retrieve data from any table owned by any user in the database.
INSERT ANY TABLE	Allows the user to add new rows to any table owned by any user in the database.
UPDATE ANY TABLE	Allows the user to update rows in any table owned by any user in the database.
DELETE ANY TABLE	Allows the user to delete rows from any table owned by any user in the database. This privilege is needed to truncate tables owned by other users.
FORCE ANY TRANSACTION	Allows the user to commit or roll back distributed database transactions related to any user in the database.
CREATE ANY TRIGGER	Allows the user to create a trigger and assign ownership to any user in the database.
ALTER ANY TRIGGER	Allows the user to modify (enable, disable, or recompile) any trigger owned by any user in the database.
DROP ANY TRIGGER	Allows the user to drop any trigger owned by any user in the database.

<i>Privilege</i>	<i>Description</i>
CREATE ANY VIEW	Allows the user to create a view and assign ownership to any user in the database. The creator must also have ALTER ANY TABLE, BACKUP ANY TABLE, DROP ANY TABLE, LOCK ANY TABLE, COMMENT ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, or DELETE ANY TABLE, depending on the exact view that the user is trying to create. These prerequisites come into play only when you are trying to access a database table for which you do not have a specific access granted to your user ID. If you have specific access grants to your user ID or the table is available to PUBLIC, you do not need the prerequisite privileges described.
DROP ANY VIEW	Allows the user to drop any view owned by any user in the database.

Privileges such as CREATE ANY VIEW and CREATE ANY PROCEDURE include a series of privileges that I qualify as being needed, depending on what the user is trying to do. If the user whom you are trying to assign the object to does not have another object with the same name as the object that you are trying to create, you do not need any of these special privileges. However, if you are trying to create a table with the same name as one that the user already has, you need to have DROP ANY TABLE to get rid of the existing table so that you can replace it with the new one.

THE DATABASE MAINTENANCE PRIVILEGES

Next on the list of system privileges are those that you use to maintain the database itself. These are some of the traditional DBA privileges that you use to create the database, add tablespaces, and so forth (see Table 11.4).

TABLE 11.4. THE DATABASE MAINTENANCE PRIVILEGES.

<i>Privilege</i>	<i>Description</i>
ALTER DATABASE	Allows the alter database command to be issued. Some of the functions of this command include mounting and opening the database, managing log files and control files, and changing the archivelog status.
CREATE PROFILE	Allows the user to create profiles that set limits on the usage of certain Oracle resources by users assigned that profile.
ALTER PROFILE	Allows the user to modify profiles.
DROP PROFILE	Allows the user to drop profiles.
ALTER RESOURCE COST	Allows the user to change the cost assigned to various resources when these costs are tracked by the database.
CREATE PUBLIC DATABASE LINK	Allows the user to create a link to another Oracle instance that is accessible by all users in the system.
DROP PUBLIC DATABASE LINK	Allows the user to drop a public database link.
CREATE ROLE	Allows the user to create a role.
CREATE ROLLBACK SEGMENT	Allows the user to create rollback segments within a tablespace.
ALTER ROLLBACK SEGMENT	Allows the user to modify the structure of existing rollback segments.
DROP ROLLBACK SEGMENT	Allows the user to drop a rollback segment.
ALTER SYSTEM	Allows the user to issue the ALTER SYSTEM command. This command is used to switch log files, check data files, set certain system parameters, kill user connections to the Oracle instance, and other similar functions.

<i>Privilege</i>	<i>Description</i>
CREATE TABLESPACE	Allows the user to create a new tablespace. Note that the Oracle user ID needs to have operating system permission and that directory. There also must be sufficient disk space available.
ALTER TABLESPACE	Allows the user to modify the configuration of existing tablespaces. This includes adding data files to the tablespace.
MANAGE TABLESPACE	Allows the user to perform warm backups of the tablespace and switch the tablespace offline and online.
BECOME USER	Switch to become another user in the database. Used for full database imports and exports only. It does not work from the SQL*Plus prompt.
ALTER USER	Change the configuration of any Oracle user account (including changing passwords).
DROP USER	Remove a user from the system.

THE MONITORING PRIVILEGE

Finally, there is a privilege used to monitor activities related to the auditing utilities that Oracle uses (see Table 11.5). (Actually, there are two, but the AUDIT ANY privilege, which enables you to audit access to database objects of any users, is covered under the “any” privileges.)

TABLE 11.5. THE MONITORING PRIVILEGE.

<i>Privilege</i>	<i>Description</i>
AUDIT SYSTEM	Allows the user to issue audit commands related to the usage of certain SQL statements, especially those related to system privileges.

TYPICAL PRIVILEGE SETS

The previous section provides a description of all the Oracle system privileges. The next challenge is to fuse the world of possible privileges that into several sets of privileges that you want to assign to your users. Under Oracle 6, you have only three system privileges (CONNECT, RESOURCE, and DBA) that are pretty self-explanatory. Under Oracle 7, it takes an understanding of your user needs and security requirements plus a little experience to form the proper privilege sets. I cannot guess at your users or security requirements; however, I can provide insight into what I have seen as typical privilege sets. Consider these when you make up your plans.

The first example to consider is a relatively simple data warehouse containing information on sales that have been made. The sales data is downloaded nightly and stored in a series of basic sales tables and a series of precalculated aggregate tables. In this situation, the following privilege sets are applicable:

- ♦ A “dummy” user ID owns all the tables and is used to execute all the data population scripts. This account is the only account that can create database objects. Because it is an account controlled by the DBA, it has a number of the “any” privileges to speed along work.
- ♦ The rest of the users on the system have the connect role and its associated privileges.

Consider next a development instance. In this instance, security is important for a common set of test tables. However, when you have developers who are trying out new table structures, you want them to create them in their own schemas to ensure that they do not impact the others until the new tables and software are ready. In this situation, you may consider the following privilege sets:

- ♦ A dummy account to own all the common database test objects. This ensures that all developers will be testing off of the same tables and these tables contain a controlled data set. This account has privileges to create all the common database objects and also SELECT ANY privileges to copy tables owned by the developers when needed.
- ♦ A developer role that allows the developers to create any of the database objects in their own schemas. They utilize private synonyms to rename their personal tables to the names of the common tables owned by the dummy user. Because private synonyms override the public synonyms used to access these tables, any software they write using the synonyms will work when placed in production.

Finally, consider an imaginary personnel system. These systems are among the most sensitive in certain organizations, especially with the many regulations

concerning privacy. This fictitious company is so concerned about security that it has appointed a separate security administrator who is responsible for owning and maintaining the tables in the system. The database administrators and operators are not permitted access to these tables. This configuration may contain the following privilege sets:

- ◆ A special DBA role is created. Most of the database maintenance privileges (CREATE TABLESPACE, ALTER DATABASE, and many others) are there, but most of the “any” privileges that relate to database objects are missing (for example, SELECT ANY TABLE).
- ◆ An operator role that has only the privileges needed to start, stop, and back up the system.
- ◆ The security administrator role that has auditing privileges and a few of the “any” privileges related to creating and dropping tables. This is normally not a problem because the security administrator is the person who has access to the dummy account used to create all the database objects.

THE “ANY” PRIVILEGE SETS

As mentioned earlier, the “any” privileges are something that I use every day. Many of them are not particularly harmful. SELECT ANY TABLE may pull in a lot of data, but nothing is destroyed if you make a typing mistake. Many of them I have never used (ALTER ANY TRIGGER, for example). For Oracle 6 users, even though you do not explicitly have these system privileges as separate entities, you have most of them when you use the DBA system privilege. Let’s go over some guidelines that I use when working with these “any” privileges using my DBA account:

- ◆ When deleting database objects, always use the fully qualified name and do not rely on synonyms. It takes only a few extra characters to specify *owner.table_name*. When you forget exactly what that public synonym is pointing to, it can save some heartburn.
- ◆ Use dummy accounts to create the objects that you want controlled in the database. You can do it using your DBA account with its CREATE ANY TABLE privilege, but remember, you need to access this account to perform the object privilege grants anyway. An advantage to coding your object creation scripts with the *owner.table_name* format is that if you are logged into the wrong account when you execute them, it tells you about your mistake. Yes, I have occasionally built a few dozen tables only to realize that I was distracted and put them in the wrong schema.
- ◆ If your data is particularly sensitive to mistakes and you have other functions to perform such as development, consider making a less-privileged

account for yourself. Utilize your personal DBA account only when needed and that will minimize the chance that your “any” privileges will do something that you do not want. Also avoid using the SYS and SYSTEM accounts whenever possible because they own too many sensitive objects that you might overwrite.

SUMMARY

This chapter provides an overview of the Oracle privilege (that is, security) scheme and the system privilege sets used to control what users are able to do with the database itself. The next chapter presents the object privileges, which control access to individual tables, views, and so forth. It is important to remember the distinctions between Oracle 6 and 7 system privileges. Oracle significantly upgraded (those who do not need security may say it *complicated*) the system privilege scheme in Oracle 7. This allows a very fine degree of control and support for a strong security administrator role, if one is desired.



- Overview
 - Object Privileges
 - Using Dummy Object Owners
 - A Typical Privilege Scheme
-

CHAPTER 12



Oracle Object Privileges

The last chapter covered the first half of the Oracle security scheme—the system privileges. These privileges control what the user is allowed to do with the system itself (create tables, for example). This chapter focuses on the other half of the Oracle security scheme—the object privileges. These privileges control what users are allowed to do with individual tables, views, and other objects.

The good news is that there are fewer object privileges than there are system privileges—only eight. The bad news is that although a user has only one set of system privileges, that user needs object privilege sets for every database object of interest. When you consider the number of users and tables that a large database may have, you understand why Oracle 7's roles become so important. It is easier to grant access to 50 tables to 3 roles than it is to grant access to those 50 tables to 200 users.

This chapter presents what the privileges are and which are applicable to which objects. Some are obvious. You can execute a procedure but not a table. Some are not so obvious. You may wonder why you can't just alter a view to which you want to add one column. This chapter reviews both of these points from the point of view of the DBA.

OVERVIEW

Every object in the database has a set of access privileges associated with it. Some are implied by the type of object. Public synonyms, for example, are universally accessible by their very definition. However, most objects have some form of protection that the owners and database administrators need to keep track of as part of the overall security schema. That is what this chapter is all about.

The first subject to master is the list of privileges provided by Oracle. All these privileges existed in Oracle 6 with the exception of execute (stored procedures did not exist in Oracle 6). Here are the Oracle object privileges:

- ♦ ALTER allows the user to modify the internal structure of the object. For example, it allows the user to add columns to a table. It does not allow the user to modify the contents of the data itself (see update, insert, and delete). Note that you cannot drop columns from a database object; you can only add or modify them.
- ♦ DELETE allows the user to remove one or more rows of data from the object.
- ♦ EXECUTE allows the user to execute a stored package, procedure, or function.

- ◆ **INDEX** allows the user to have access to the data in a table to create an index associated with the table. Note that the select privilege is not enough for a user to create an index on the table.
- ◆ **INSERT** allows the user to add rows of data to the object.
- ◆ **REFERENCES** allows the user to create or alter another table that references this table as a foreign key. It does not allow the user granted this privilege to alter this table in any way. Instead, it grants that user permission to create a new table, which has a rule that the contents of one or more of its columns has to match a corresponding column in the table on which the REFERENCES privilege is being granted (or else be null).
- ◆ **SELECT** allows the user to read rows from the object using the SQL select statement. This is probably the most common privilege.
- ◆ **UPDATE** allows the user to modify the contents of existing rows in the table by using the update command.

The owner of the object has the full set of privileges available on that object when the object is created. These privileges can be given up via the `revoke` statement, but usually the owner can do anything that is necessary. (The actual statements used to grant and revoke privileges are discussed in Chapter 13.)

Figure 12.1 shows how the object privileges apply to the various objects within the database. There is some overlap between system and object privileges. Some objects within the database require system privileges for creation and others require object privileges. Some objects use object privileges to restrict access and others have internal protection mechanisms (see Table 12.1).

Figure 12.1.
Mapping object
privileges to objects.

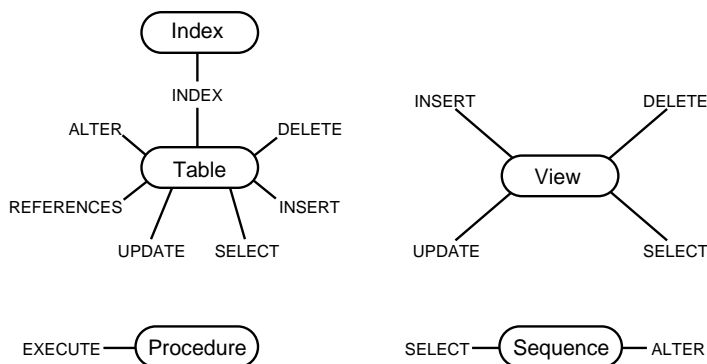


TABLE 12.1. PROTECTION MECHANISMS FOR DATABASE OBJECTS.

<i>Object</i>	<i>Protections</i>
Database	System privilege ALTER DATABASE.
Tablespace	System privileges CREATE TABLESPACE, ALTER TABLESPACE, MANAGE TABLESPACE, DROP TABLESPACE.
Tables	System privileges CREATE TABLE, CREATE ANY TABLE, ALTER ANY TABLE, and DROP ANY TABLE protect the table itself. Object privileges ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT, and UPDATE protect the data within the table. System privileges SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, and DELETE ANY TABLE can be used to override the object privileges.
Indexes	The system privilege CREATE TABLE gives the user the privilege to create indexes. The object privilege INDEX is required to access the data in a table to build the index. The system privileges CREATE ANY INDEX, ALTER ANY INDEX, and DROP ANY INDEX can override the object privileges.
Views	The system privileges CREATE VIEW, CREATE ANY VIEW, and DROP ANY VIEW protect the object itself. Object privileges DELETE, INSERT, SELECT, and UPDATE protect the data within the view.
Sequences	The system privilege CREATE SEQUENCE protects the object itself. The object privileges ALTER and SELECT protect the data in the sequence. The system privileges CREATE ANY SEQUENCE, ALTER ANY SEQUENCE, DROP ANY SEQUENCE, and SELECT ANY SEQUENCE can override the object privileges.
Procedures, and so on	The system privilege CREATE PROCEDURE protects the object itself. The object privilege EXECUTE protects access to the software. The system privileges CREATE ANY PROCEDURE, ALTER ANY PROCEDURE, DROP ANY PROCEDURE, and EXECUTE ANY PROCEDURE can override other privileges.

<i>Object</i>	<i>Protections</i>
Synonyms	The system privilege CREATE SYNONYM protects the object itself. Private synonyms are accessible only by their owners and do not have object privileges. The system privileges CREATE ANY SYNONYM and DROP ANY SYNONYM can be used to override other privileges.
Public Synonyms	The system privileges CREATE PUBLIC SYNONYM and DROP PUBLIC SYNONYM protect the object itself. All public synonyms are accessible by all users and therefore there are no object privileges associated with public synonyms.

Figure 12.2 illustrates the protection of the objects themselves, and Figure 12.3 shows the protection of the data within the object.

Figure 12.2.
Protection of the objects themselves.

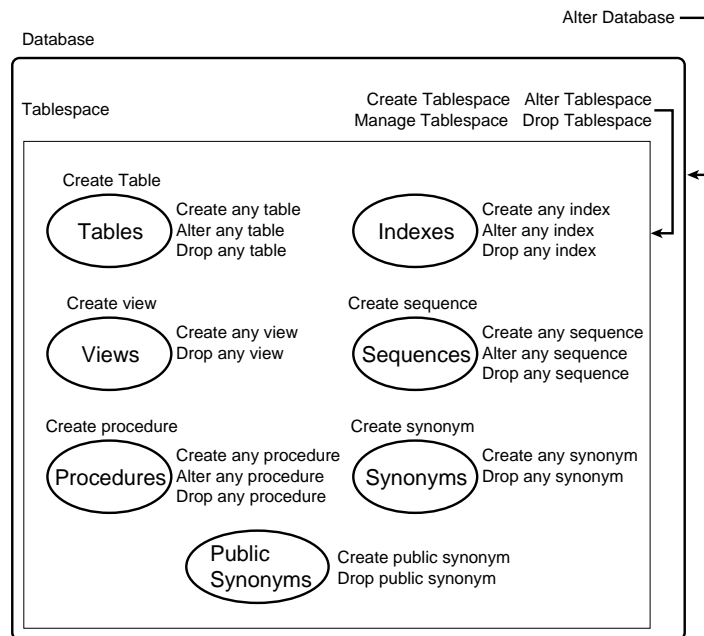
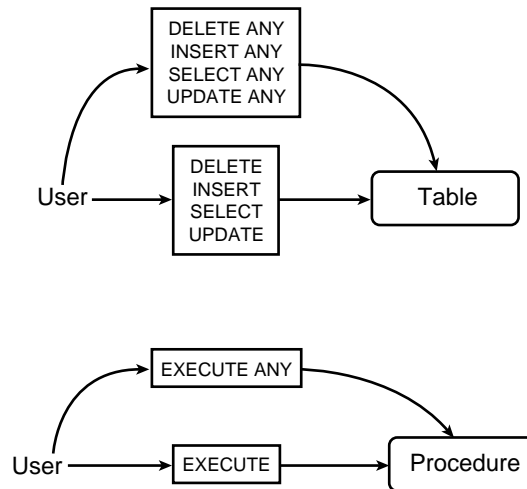


Figure 12.3.
*Protection of the data
within the object.*



You may feel a little challenged right now with all the things to think about while devising a scheme to protect your data. Let's add a few more considerations to the challenge. You may have a table to which you have granted access to no one other than yourself. However, any user who has the `SELECT ANY TABLE` or, worse yet, `DELETE ANY TABLE` privilege has access to that table. This comes into play when you have a number of public synonyms and use generic table names (for example, `test`, `names`).

Another thing to consider is Oracle's capability of restricting access down to the column level on tables. I have never used it for fear of the performance impacts of having to check access on a column-by-column basis, but it is a tool that you can use when it makes sense. Basically, you can give users an object access (`SELECT`) on only certain columns within the table. It would be a tough system to administer, but it could be necessary.

OBJECT PRIVILEGES

That last section presented a lot of challenging concepts at a rather rapid pace. This section reduces the pace a bit and explores the various object privileges that you can grant on an object-by-object, privilege-by-privilege basis.

The first group to start with is the object privileges associated with tables. These are the fundamental data storage devices and the type of object privilege grants that make up the vast majority of most systems. These are the privileges that you can grant on tables:

- ◆ ALTER allows the user to change the structure of the table. This includes adding and deleting columns and changing the storage parameters associated with the table.
- ◆ DELETE allows the user to delete rows of data from the table. It does not allow the user to delete the table itself.
- ◆ INDEX allows the user to access the data in the table to create an index. This can be important in that Oracle automatically maintains indexes on its tables. Therefore, if a large number of users is creating indexes on tables, your data entry performance will suffer. One interesting feature of this privilege is that it cannot be granted to a role. It can be granted only to individual users.
- ◆ INSERT allows the user to add new rows of data to the table.
- ◆ REFERENCES allows the user to use the table as a foreign key in another table. This basically links the data in the two tables via one or more columns in each table. This is the other table object privilege that cannot be granted to a role. It can be granted only to individual users.
- ◆ SELECT allows the user to read information from the table. This is the most common privilege granted to the user community. It may be the only privilege granted to users in data warehouses.
- ◆ UPDATE allows the user to change the contents of existing rows of data in the table. It does not allow new rows to be created or existing rows to be deleted.

The next objects that have privileges associated with them are the views. Note that users have access to data from a table through a view even though they do not have access to the base table. This technique can be used to limit the amount of information that certain users can work with, thereby eliminating the need to use column-level access privileges. When a view is used to join columns from multiple tables together, the creator needs the appropriate access to all the tables included in the view to create the view. Note that you cannot alter the structure of a view. Because it does not contain any data itself, the designers of Oracle felt that it was easier to drop and re-create a view than to allow modification of the view. Therefore, they enable you to issue a command to **CREATE OR REPLACE VIEW** (which enables you to keep any privileges or grants that you may have made while re-creating the view). The following are the object privileges associated with views:

- ◆ DELETE allows the user to remove rows of data using the view. This can be very tricky because you may not want to delete the rows of data from both tables when you have the two joined together. Try to avoid allowing users to delete data using a view. DELETE does not work for views that contain data from multiple tables.

- ◆ INSERT allows the user to add rows of data using the view. Again, this can be somewhat tricky because there may be mandatory fields in the base table that are not included in the view (for example, primary keys). INSERT does not work for views that contain data from multiple tables.
- ◆ SELECT allows the user to read data using the view. This is the “safe” view privilege because all the user looks at is the data.
- ◆ UPDATE allows the user to modify existing data using the view. Allowing the user to modify such things as primary keys should be avoided in views. UPDATE does not work for views that contain data from multiple tables.

Another database object that has its own set of object privileges is the sequence. These are relatively simple objects (a list of numbers). There are only two object privileges associated with sequences:

- ◆ ALTER allows the user to modify the structure of the sequence.
- ◆ SELECT allows the user to read the next value from the sequence.

The final database object with object privileges is the procedure. This group includes packages (individual PL/SQL routines) and functions (which are similar to packages, but have slightly different calling parameters). It does not include triggers, which are bits of software that are automatically tied to tables. Basically, you can force that software every time any user performs a specified action (for example, an update) on that table. These are an Oracle 7 feature (one less thing for Oracle 6 DBAs to worry about!). The procedure has only one object privilege applicable to it—EXECUTE. This allows a user to run the software contained within the procedure.

A final note on the column privileges capability of Oracle: As mentioned earlier, I have never used them because of the overhead associated with having to check column privileges in addition to table privileges in a query. I usually tend to use views for this function, but there are applications wherein column privileges are the only practical answer. The good news is that column privileges are relatively simple to grant and revoke. You use all the privileges that you are familiar with for tables (for example, SELECT) and merely put the names of the applicable columns in parentheses after the table name. You can even work up some interesting schemes whereby, if you want to grant select on all but one column, you grant select to the whole table to a user and then revoke select for that one column.

USING DUMMY OBJECT OWNERS

The previous sections present the list of privileges associated with the various types of database objects. The next couple of sections tie all these concepts of system and object privileges together. You’ll see several examples that represent environments

that I have come across in my travels. These may not match your individual needs exactly, but you should be able to take these concepts and adapt them slightly for your needs.

The first concept is that of the dummy user. We all have users who just do not seem to be able to grasp the concepts of computers, but I would not call them dummies. We in the computer industry are not known for our understanding of their business functions, and let's hope that they are not calling us dummies when they write accounting books. Recall from the last chapter that a dummy user is an Oracle user ID that is not associated with a person. It exists strictly to own objects within the database. There are several advantages to this scheme:

- ◆ The password to this account can be restricted to DBAs or senior developers. This enables you to control precisely who can perform major modifications to these critical database objects, especially the more serious functions such as DROP TABLE.
- ◆ You or some senior developers need access to this account to perform the object privilege grants on the database objects.
- ◆ Because it is not often possible (due to inadequate space to copy large tables) to transfer database objects when the original owners leave the project and their accounts need to be deleted, these dummy owners are always with the project.
- ◆ No developers can accidentally delete a key production table when they type in an incorrect table name or forget that they are working in the production instance versus the test instance. The dummy tables are restricted to key individuals and used only for object creation and deletion purposes. Some of you will not allow any developers at all into the production instance with anything other than connect privileges; but for others of us this is a real concern.

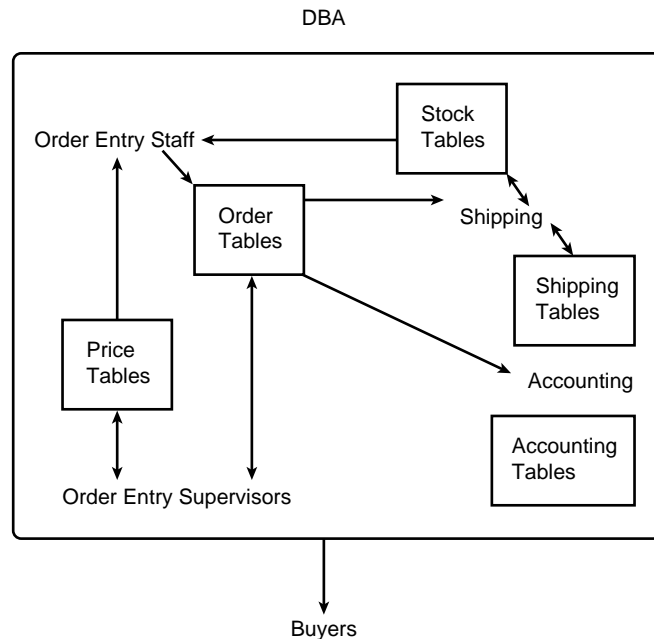
A TYPICAL PRIVILEGE SCHEME

This section sketches out some typical privilege schemes that are a combination of what I have seen in several instances during my travels. Consider whether any of them could be used as a starting place for your instance. For this discussion, you explore four privilege schemes:

- ◆ An order entry system
- ◆ A data warehouse system
- ◆ A data warehouse development instance
- ◆ A laboratory or research environment

The first environment is an order entry system (see Figure 12.4). In these systems, you typically have a group that enters orders into the system. This data is stored and forwarded to groups involved with accounting/billing functions and shipping. Most of the order entry staff is permitted to enter new orders, but supervisor permission is required to change prices and other such functions. Finally, there are often groups, such as buyers, who monitor what is selling and what is in stock to ensure that the correct products are being built and acquired.

Figure 12.4.
A sample order entry environment.



The exact details of how this works will vary between companies, but let's divide the users into six roles:

- ♦ DBA: (this is a book on database administration, I would not forget to include us).
- ♦ order_entry: staff involved with taking the orders and entering the data into the system.
- ♦ order_supervisor: supervisors in the order entry process who are allowed to update certain data fields that the order_entry staff is not permitted to modify.
- ♦ accounting: these people perform credit checks, billing, and so forth, and update the tables associated with accounting data.

- ◆ shipping: these people put the components of the order together and ship it out. They also update the shipping data tables.
- ◆ buyers: these people are allowed to monitor the flow of goods and orders, but not to change anything. They are strictly observers in this process.

Now let's divide the many user data tables in the instance into five categories—order, shipping, stock, accounting, and pricing. Let's also assume that there is a dummy user account called “oe” (for order entry) that owns all the user data tables. Based on the scenario described, and giving some leeway to fill in the blanks because this is a fictitious company, I would propose the following schedule of privileges:

<i>Group</i>	<i>Privileges</i>
DBA	Default Oracle 7 DBA privileges. This gives DBAs full access to the database.
order_entry	CREATE SESSION, INSERT on the order tables. SELECT on the shipping tables (provide order status to customers). SELECT on stock tables (see whether in stock). No access to accounting tables. SELECT on pricing tables (quote prices).
order_supervisor	CREATE SESSION, SELECT, INSERT, UPDATE, DELETE on order tables. SELECT on shipping tables. SELECT on stock tables. SELECT on accounting tables. SELECT, UPDATE on pricing tables.
accounting	CREATE SESSION, SELECT on order tables. No access to shipping or stock tables. SELECT, INSERT, UPDATE, DELETE on accounting tables. SELECT on pricing tables
shipping	CREATE SESSION, SELECT on orders and accepting tables (see credit check was OK). SELECT, INSERT and UPDATE on stock and shipping tables. No access to pricing tables
buyers	CREATE SESSION, SELECT on all the tables.

Notice in this example that all the users in the system are covered, and you did not have to give out the DROP ANY TABLE or DELETE ANY TABLE privileges. These nasty and powerful privileges are usually used only by DBAs and, in some cases, senior developers. You can have a fully functional instance with everyone limited to what they need to get their jobs done.

An environment that contrasts sharply with that of the order entry system is the data warehouse. The data warehouse concept basically states that you make a copy of data stored in an operational system, such as the order entry system discussed in Chapter 11, and store it in a separate database instance. You often summarize the data and store the summary information in new tables that promote rapid answers to common questions.

Consider the situation where the buyers in the order entry system start to work very hard to analyze their product mix, distribution of purchases throughout the year, and other factors so that they always know exactly what was selling at the time. The system administrators and order entry staff notice that the order entry system is slowing down significantly when these buyers run their analyses. The solution chosen is to purchase a new computer with Oracle and transfer the data from order entry to this new Oracle instance nightly for buyer analysis.

Now management throws in a slight wrinkle into this system. They decree that all purchases will be categorized according to the four product groups sold by this company. They do not want buyers to be able to access data that does not fall into their category. The decision is made to split all orders coming down from the order entry system into four sets of tables, each set representing a product group (for simplicity, no order will cross two product groups). All the objects are owned by the dummy user known as “warehouse.” All the nightly data update jobs are run using this user ID. Therefore, all that is needed is to create four roles. Each role will have `CREATE SESSION` and `SELECT` privileges on the tables that correspond to a particular product group. It’s a simple and effective setup. No one but the DBAs and the dummy user can update anything.

Development instances present a series of challenges that are not found in production instances. In production, it is easy to restrict users to all but the most basic privileges. Application developers need to have enough power to get their work done, but not enough power to do any real damage. For the next example, let’s choose the Oracle instance used to develop the data warehouse discussed in the last section. This company is wise enough to use a separate instance for development and testing (I recommend it whenever possible, but then again I don’t see the fun of playing Russian roulette either). In this instance, I would recommend the following roles and privilege sets:

- ♦ The default DBA role (of course).
- ♦ Each of the four roles used in production. You should also create dummy users who each has one of these roles. This enables you to test your applications with the same security scheme that is used in production. It would not be valid to test an application as the table owner and expect it to work the same for a different user account.

- ◆ A developer role. Assign the following privileges to this role:

CREATE SESSION

ALTER SESSION

CREATE TABLE

ALTER TABLE

CREATE VIEW

CREATE PROCEDURE

CREATE SYNONYM (not CREATE PUBLIC SYNONYM)

CREATE SEQUENCE

CREATE TRIGGER (perhaps, if they are used)

Again, this is a rather simple scheme. It avoids using the nasty “any” privileges and still should enable the developers to get their jobs done. I would like to emphasize the importance of testing new software with accounts that will have the same security scheme as the end users. As mentioned earlier, I once received an application that was tested with operating system accounts that, in effect, had system administrator privileges and Oracle accounts that had full DBA privileges. Surprisingly, nothing worked when the application was transferred to the real world where users were just users, not DBAs. Trust me, testing using only developer or object owner Oracle IDs is inviting the opportunity for you to shoot yourself in the foot.

Finally, on the other end of the spectrum from most of the systems listed previously is the laboratory or research system. In this case, every group wants to be able to create and freely manipulate its own data. I always suggest setting up groups for each project team so that they can share data with one another freely (that is, grant to the group as opposed to giving grants to all the members of that group individually). I suggest giving each of these groups the same privilege sets that were given to the developers in the test data warehouse system. Of course, there may be lab environments (for example, pharmaceutical testing) that need to exercise stringent controls over lab data. However, this example covers a number of installations wherein the database is a general-purpose tool to store data as opposed to a repository of controlled corporate business application data.

SUMMARY

This concludes the discussion of the Oracle privileges and the object privileges specifically. Under Oracle 6, the system privilege scheme is simple, which can be either a blessing to the DBA or a limitation on the ability to secure a sensitive application properly. The object privilege scheme is in place with the exception of

stored procedures, which do not exist until Oracle 7. Under Oracle 7, there is a rich scheme for protecting objects and system access. The use of roles (discussed in Chapter 13) makes administration of this system much easier.

This chapter presents the Oracle object privileges and how they apply to the various objects within the Oracle database. There are a few things to watch out for when using object privileges and some of them do not mean exactly what you might think. You should always be aware of system privileges (such as `DELETE ANY TABLE`) that can override the object privilege scheme that you have set up. The examples in this chapter should reduce your fear that a sound privilege scheme has to be extremely complex. Most of the system privileges are not given to average users or even developers—they are just too powerful.

The next chapter presents more discussion on roles in Oracle 7, including the commands used to grant and revoke access to the Oracle system. This will complete the picture of privileges and access control in Oracle instances. There are a few finer tricks that can be used (for example, using quotas to limit the amount of space and which tablespaces a developer can use) that are discussed later.

-

Roles and Grants

The last several chapters may have been tough for the more action-oriented types. You had to learn *what* you have control over before you could see *how* you control them. This chapter is designed to address these how-to topics. Again, it is impossible to guess what roles and privilege sets are best for your individual situation. Therefore, this chapter discusses the commands and concepts needed to build a privilege scheme and then shows several examples. You can then use these examples to build your own privilege schemes.

The first goal is to present the concept of roles in Oracle, which greatly simplify security maintenance. Next security administration without roles, which are available only in Oracle 7, is explored. Then several typical privilege schemes that I have run across in my consulting travels are presented. The chapter winds up with one of my favorite topics—the use of scripts to capture privilege grants for future use.

Note

Oracle 7 simplifies security administration with roles. Oracle 6 makes you grant privileges to each individual user.

By the end of this chapter you should be comfortable with what it takes to assign and revoke Oracle privileges to users. You should also be able to build up a security scheme for new applications as they arrive. Of course, if you purchase some off-the-shelf applications, such as Oracle Financials, the privilege scheme may be highly determined by the vendor. However, you will know the questions to ask about their security schemes so that you can assign the correct privileges to your users.

INTRODUCTION TO GRANTS

Much as with the military, there are no rights, only privileges. Before you can do something in Oracle, you need to have the appropriate person grant you the privilege to do it. In Oracle, DBAs typically control system privileges, and object owners control access to the objects that they own. DBAs and object owners assign privileges using the SQL `grant` command. The format of this command is rather simple:

```
grant privilege to user/role;
```

The words `grant` and `to` are the easy parts. The *user* or *role* is fairly easy. You insert the name of the role or user. If you want to grant the same set of privileges to multiple users or roles, you separate these names with commas. The only remaining part of this command is the privilege itself. There are two forms of this command. For

system privileges, you merely list one or more system privileges in this part of the command. If you wish to grant multiple privileges in a single command, you separate the privileges by commas. The privileges are exactly as you saw them in Chapter 11. For privileges that contain multiple words, you separate the words with spaces. (Oracle is smart enough to figure out what you mean.) For example, to grant the system privileges `create table` and `create view` to `bsmith` and `jsmith`, you type the following:

```
grant create table,create view to bsmith,jsmith;
```

For object privileges, you need to specify both the privilege that is being granted and the object on which the privilege is being granted. If you wish to grant more than one privilege in a single statement, you separate the list of privileges by commas. You separate the list of privileges from the object on which these privileges are granted by the word `on`. As with system privileges, you can grant the privileges to one or more users, separated by commas. For example, to grant `select` and `delete` privileges on the famous `golf_scores` table to `bsmith` and `jsmith`, you type the following:

```
grant select,delete on golf_scores to bsmith,jsmith;
```

Because user needs can change over time, another important capability is being able to take privileges granted away from the users. Perhaps the users have changed jobs or you have been told to tighten up security in your database. Whatever the reason, the command to remove specific privileges from a given user is the following:

```
revoke privilege from user/role;
```

Here is an example of this command that revokes the `select` privilege granted to `bsmith` on the `golf_scores` table:

```
revoke select on golf_scores from bsmith;
```

So far, you have learned the commands needed to give and take permissions away from users on both the system as a whole and individual database objects. When a user is having access problems or when you are performing routine security audits, it is necessary to be able to determine what privileges are currently granted in the database. To do this, there are two DBA views (`dba_tab_privs` and `dba_sys_privs`) that can be queried to find this information. Views are discussed in a later chapter, but for now, think of these DBA views as objects that you can query to find out information. The first provides insight into the system privileges that have been granted to users. The second shows the table access that has been granted. The following are examples of queries to the `dba_sys_privs` and the `dba_tab_privs` views. Note that you can use `where` clauses to focus in the particular tables or users that you want:

```
SQL> select * from dba_sys_privs;
```

GRANTEE	PRIVILEGE	ADM
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE DATABASE LINK	NO
CONNECT	CREATE SEQUENCE	NO
CONNECT	CREATE SESSION	NO
CONNECT	CREATE SYNONYM	NO
CONNECT	CREATE TABLE	NO
CONNECT	CREATE VIEW	NO
DBA	ALTER ANY CLUSTER	YES
DBA	ALTER ANY INDEX	YES
DBA	ALTER ANY PROCEDURE	YES
DBA	ALTER ANY ROLE	YES
DBA	ALTER ANY SEQUENCE	YES
DBA	ALTER ANY SNAPSHOT	YES
DBA	ALTER ANY TABLE	YES
DBA	ALTER ANY TRIGGER	YES
DBA	ALTER DATABASE	YES
DBA	ALTER PROFILE	YES
DBA	ALTER RESOURCE COST	YES
DBA	ALTER ROLLBACK SEGMENT	YES
DBA	ALTER SESSION	YES

```
SQL> select * from dba_tab_privs
      2 where owner='JGREENE';
```

GRANTEE	OWNER	TABLE_NAME	
GRANTOR	PRIVILEGE		GRA
BSMITH	JGREENE	GOLF_SCORES	
JGREENE	DELETE		NO
JSMITH	JGREENE	GOLF_SCORES	
JGREENE	DELETE		NO
JSMITH	JGREENE	GOLF_SCORES	
JGREENE	SELECT		NO

```
SQL>
```

You may have noticed the `ADM` column in the output of these views. This column reflects whether this user has administrative privileges to grant this privilege to other users. For example, if you grant `select` on the `golf_scores` table to `bsmith` with the `admin` option, `bsmith` can grant `select` on `golf_scores` to other users. `bsmith` can't grant `delete` on `golf_scores` to other users unless `bsmith` has the `admin` option associated with the `delete` privilege on this table. Here is the format for granting the `admin` option to a user:

```
grant privilege to user/role with admin option;
```

You may wish to limit certain creation privileges granted to users. For example, you won't want one developer to consume every byte of space in every tablespace in your instance. A tool that you can use to limit this access is the tablespace quota. This is covered in more detail in Chapter 26 on user account maintenance. For now, just understand that you can limit the amount of space consumed by the user and also specify which tablespaces the user can use for entering data. For example, the following grants a limit of 10M in the user's tablespace to user `jsmith` (once quotas are implemented, Oracle assumes that you have a quota of zero on all tablespaces for which users have not explicitly been granted a quota, unless that user has the unlimited tablespace system privilege):

```
alter user jsmith quota 10M on users;
```

Finally, you may wish to grant certain privileges to every user in the database. Perhaps you live in a very open environment where everyone should be enabled to view all data tables. Perhaps you do not wish to have to remember to grant create session (which is needed to even connect to your Oracle instance) to every user ID in the system. Anyway, you can grant privileges to everyone by putting the word `public` in as the grantee:

```
grant create session to public;
```

INTRODUCTION TO ROLES (VERSION 7 FEATURE)

This section has no use for Oracle 6 DBAs (sorry!). Roles make the database administrator's security job much easier, but you have to have Oracle 7 to get this benefit. Let's hope that the remaining version 6 DBAs can convert their applications to version 7 soon so that they can take advantage of this and other new features.

Roles are an incredibly simple and powerful concept for the DBA. When databases were small and accessed by a limited number of individuals, it was acceptable to grant a specific access to each table that a user required. However, as databases grew and the number of users grew with them, this became a nightmare of maintenance. Operating systems have used the concepts of groups for years to give file access to individuals based on their job descriptions. In fact, many business organizations determine what access individuals have to a particular data set by their job descriptions (payroll clerk, accounts receivable manager, and so forth). The Oracle role accommodates this real-world privilege scheme by allowing the DBA to assign privileges to users based on a series of roles that the DBA can define to the database.

The concept of roles is relatively straight-forward. The real trick for the DBA is to map out a scheme of roles that enables users to get their jobs done and is maintainable. This topic is covered later in this chapter. The first step in the process is to create the roles that you need with this SQL command:

```
create role role-name;
```

For a role to be useful, it needs privileges to be granted to it and users assigned to it. After the user is created, you grant privileges to the role using the grant command discussed in the last section. You then grant access to users to the role by using that same grant command (where the role name is the privilege). For those of you familiar with object-oriented concepts, the users inherit the privileges of the role when that are assigned to the role. This is a dynamic relationship. If you assign users to roles and later change the privileges assigned to the roles, the users acquire those new privileges when they next connect to Oracle.

Perhaps a concrete example would be beneficial here. Let's create a role called "developer" for software developers. In this world, the developers are allowed to create and modify packages, views, and tables. They need the ability to connect to the system, but do not need any other special privileges. Let's also add a user with ID bsmith to the system and this group. The following example shows a sample SQL*Plus session that the DBA would run to set up this plan:

```
SQL> create user bsmith identified by brad;
```

User created.

```
SQL> create role developer;
```

Role created

```
SQL> grant create procedure,create table,create view,create session to developer;
```

Grant succeeded.

```
SQL> grant developer to bsmith;
```

Grant succeeded.

```
SQL>
```

With the basics of roles out of the way, there are a few other topics that a DBA should understand to get the most out of roles. The first is the concept of default roles. When you grant a role, it is a default role, one that users have active whenever they connect to the Oracle instance. You can establish certain user's roles as default roles. The nondefault roles require an overt act on the part of the user to obtain their privileges. To establish a specific list of default roles, you issue this command:

```
alter user userID default role list-of-roles;
```

Here is an example:

```
alter user bsmith default role manager,developer;
```

If users want to access roles other than their default roles, they have to issue a command for the format:

```
set role list-of-roles;
```

This is a somewhat tricky command. When you provide the list of roles, it does not add to the roles that you already have in place. Instead, it becomes the list of roles that are enabled. Any roles that are not in this list become disabled, including default roles. The user can still access privileges from this role by issuing another `set role` command.

If you think of a role as a bucket for privileges, it is important to understand what that bucket can contain. First, it can contain most object privileges. It can also contain system privileges. Finally, it can contain other roles. I normally do not recommend having a hierarchy of roles. It may make it easier to grant the role privileges by building this hierarchy, but you can lost track of the individual privileges granted and accidentally grant an inappropriate privilege to a certain user. However, there are DBAs who are quite comfortable with using a role hierarchy, first building fundamental roles and then building higher-level roles that are made up of the fundamental roles. In the end, you have to choose whichever method works best for you.

It is sometimes useful to apply passwords to various roles. Users will not be challenged for this password if this is one of their default roles. However, if you set it up so that this is not a default role, they will be challenged for a password when they attempt to access that role. This is a way to implement some password security for special functions even when using automatic login accounts (OPS\$), which are explained later when you explore user administration. The following is an example of a role that is created with a password:

```
create role developer identified by birdseye;
```

If you can create something, you may also wish to remove what you have created. Perhaps there have been major changes to the application and a role is being split into several roles, each with privileges in different areas of the application. You want to create the new roles and then drop the old role that had all the privileges. The command to perform this is the `drop role` command:

```
SQL> drop role developer;
```

```
Role dropped.
```

```
SQL>
```

Remember that you can still grant privileges to individual users when you are using roles. For example, if there is a common privilege set for all finance department managers with the exception of Susan, who can also define the list of valid budget categories, you can assign all the basic privileges to the `finance_manager` role and assign all finance department managers including Susan to this role. You can then perform separate grants to Susan to perform her special job functions. I always prefer to keep things simple, but if you have a special need, remember that this capability is available.

GRANTS WITHOUT ROLES

For those of you who have Oracle 6 or just do not want to use roles, there is the old-fashioned way of granting every privilege to each individual user. You have to grant the appropriate system privileges to each user. You also have to grant the appropriate object privileges to each user. If you have an average user with five system privileges and five access privileges on 10 tables, you would have a total of 45 privileges to grant per user. A system with just 20 users would have 900 individual grants. Such a system would include grants such as the following:

```
grant select,insert on golf_scores to jsmith;  
grant select,insert on golf_scores to bsmith;  
grant select on golf_courses to jsmith;  
grant select on golf_courses to bsmith;
```

and so forth. There are a few things that you can do to make this easier. First, build scripts for each type of user that enable you to input a user name and the script, then performs all the grants that are appropriate to that type of user. This can be thought of as implementing roles through scripts. Another way to make things easier is to grant as many privileges as possible to public. You do not want to compromise security, but if it is a privilege that everyone has, save yourself some time.

A TYPICAL PRIVILEGE SCHEME

There probably is no such thing as a single typical privilege scheme. Instead, let's look at several examples. The details do not matter as much as understanding the differences between the examples and the concepts behind them. If you capture this knowledge, you can create the privilege scheme that works for your environment. The keys to implementing this scheme are a sound knowledge of the Oracle privileges and an understanding of your user and application needs. Chapters 11 and 12 should serve as a foundation for Oracle privilege knowledge. The understanding of your user and application needs are left to the student as an exercise.

Let's examine three typical privilege schemes that I have run across in my adventures in consulting. The first scheme is a typical online transaction processing system where different people can enter different bits of data. The second scheme is the data warehouse where almost everyone accesses data and only a handful of accounts can actually write data. The last scheme is a sample security scheme for a development Oracle instance where there are some developers who are trusted with more power than others.

The first is a fictitious accounting general ledger system. You research your user needs (or have the developers do this for you before you allow them to transition the application to production), and determine that the following rules apply to this system:

- ◆ You have clerks who can enter transactions. They are not allowed to modify any data or control the lists of valid accounts or legal values. If they make a mistake, they have to make a correcting entry.
- ◆ You have supervisors who can enter transactions and modify the lists of valid accounts and legal values. Not even supervisors can modify the data in the general ledger itself.
- ◆ Data is downloaded nightly from electronic data interfaces with other vendors and customers. This data needs to be entered automatically by nightly batch processing routines. Again, this processing adds new records, it does not allow existing data to be updated or deleted.

Many of you may work in similar environments. Perhaps you are wondering why a few obvious considerations for security have not been mentioned. Perhaps you have some personal opinions as to how these things should be set up based on problems that you have run across in the past. These are valid points to consider. Businesses are not homogenous. Every organization adapts its business processes to its particular business area, corporate culture, region, and country. A good application adapts to the business needs. These security rules are the part of the application that ensures that data is properly accessed and should be thought of in that light.

Based on the rules previously described, I would create the following roles in this database:

- ◆ DBA
- ◆ gen_led (an object owner account not associated with a person)
- ◆ clerk
- ◆ supervisor
- ◆ EDI

Once the roles are established, it is time to grant system and object privileges to these roles. For simplicity, let's group all the application tables into two groups—transactions and valid values. I have found that for transaction processing systems, as opposed to data warehouses and development environments, the main goal of security is to control who can do what with various types of data (insert, update, select, and delete). Of course, this would all have to be reviewed with users because they usually do not tell you all of the rules in your initial discussions. However, from the data I have presented, I would construct a role scheme as follows (see Figure 13.1).

- ♦ DBA: the standard privileges granted to the default DBA role in Oracle version 6 or 7.
- ♦ gen_led: an object owner account. Only the DBA or project manager has access to this account to add new tables, modify existing tables, and provide grants to these database objects. These are the privileges for this account:
CREATE SESSION (an easy one to forget, but is needed)
CREATE TABLE
CREATE PROCEDURE
CREATE PUBLIC SYNONYM
DROP PUBLIC SYNONYM
CREATE SEQUENCE
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
unlimited tablespace (possibly, if used by the DBA)
- ♦ clerk: the first line of users of the system. Their job is defined to enter data. For control reasons, you do not overwrite existing data (update or delete). Instead, you make a correcting entry that shows an audit trail. They are not permitted to modify the valid values tables, only the transaction tables. The following is a good scheme for the clerk role (or users if you have version 6):
SELECT, INSERT on transaction tables
SELECT on valid values tables
CREATE SESSION

- ◆ supervisor: the supervisors in the accounting general ledger group are the highest level of users who interact with this system. In many locations they are allowed to modify transaction data; however, in this instance, they cannot because having a detailed audit trail of all entries made (even bad ones) is important for financial control. The supervisors are allowed to modify the valid value data (including adding new rows, modifying existing rows, and deleting rows):

CREATE SESSION

SELECT, INSERT on transaction tables

SELECT, INSERT, UPDATE, DELETE on valid values tables

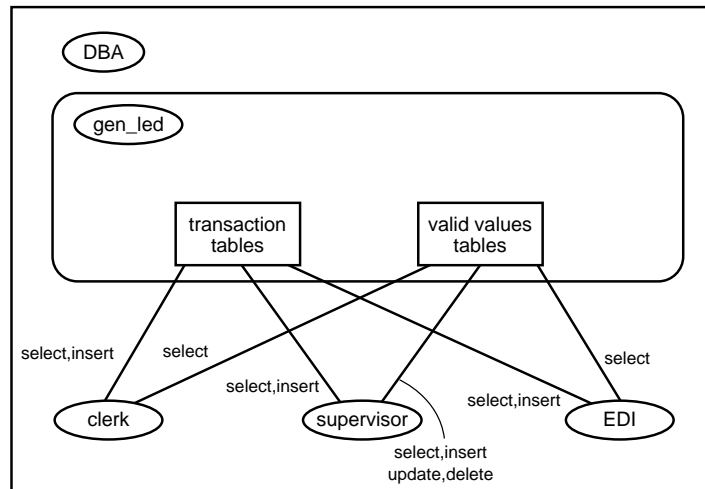
- ◆ EDI: this is a fictitious account that is used to record nightly data downloads from and prepare data uploads to other companies. Notice that even though these uploads and downloads are done through batch scripts run at night, the CREATE SESSION privilege is still needed to connect to the Oracle instance. Note that the following scheme is almost identical to the clerk role. A separate role is used for this radically different function to deal better with changes in privileges that may come up in the future (for example, users allowed to modify certain tables). This may be a consideration when you plan out your security scheme:

CREATE SESSION

SELECT, INSERT on transaction tables

SELECT on valid values tables

*Figure 13.1.
Sample transaction
processing system
privilege scheme.*



The first example provided the methodology for developing a security scheme for a new application. I tend to do very few tasks from scratch. Instead, I like to get out reference books and look at scripts that I have created before to use as examples in my current task. For those of you who think in a similar manner, the next example is a security scheme for a very common Oracle application today—the data warehouse. In the typical data warehouse scenario, you have an online or batch transaction processing system (often on a mainframe computer) that serves as the source of data for the warehouse. On some routine interval, you receive data from the transaction processing system. I have usually seen nightly downloads, but you could do it hourly, weekly, or whatever interval fits your business needs.

The key difference in a data warehouse is that most of the security rules related to who can update, delete, or insert are not applicable to the user accounts. There are controlled download routines that are run using special Oracle accounts used only for download, or the privileges required by the download scripts are granted to the system operators. The only restrictions for the users concern what data they are allowed to read.

For this example, let's consider a system that stores information related to orders downloaded from a transaction processing system. The orders themselves are stored in tables that are not directly accessed by the users. Instead, scripts are run that extract and summarize key data elements into tables designed to answer the questions of production, sales, and shipping. The common users in each of these groups are not allowed to access the data in the other areas. Certain corporate planners are permitted to access data in all systems to provide overall performance and planning reports. Assuming that the rules that I have presented here are correct and complete, I would propose the following privilege scheme for this data warehouse (see Figure 13.2):

- ♦ production: these users are concerned with assessing the impact of orders on what the company is producing. They divert resources, acquire raw materials, and so forth, based on the orders that are placed. They require the following privileges:

CREATE SESSION

SELECT on production tables

- ♦ sales: these users are concerned with fulfilling sales quotas, profit margins, pricing, and similar things. They use the data to plan where to market the products and the price at which to market the products. They require the following privileges:

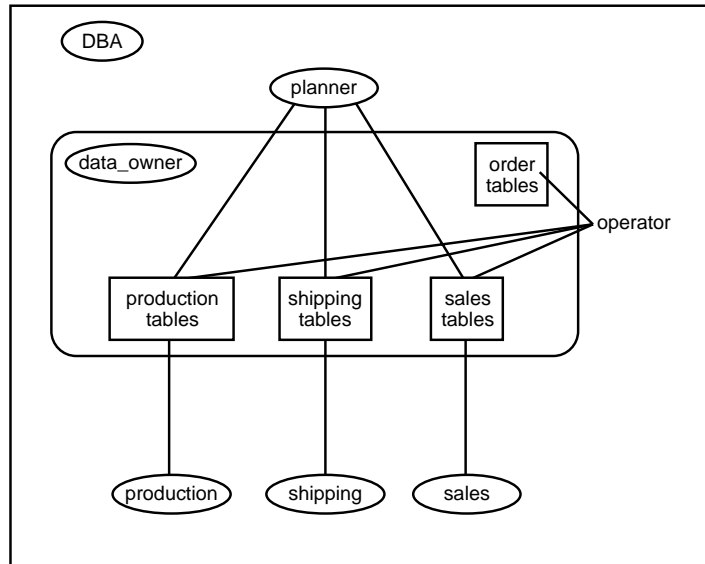
CREATE SESSION

SELECT on the sales tables

- ◆ shipping: these users are concerned with planning what items will be shipped to what locations. They arrange trucks and so forth for the products. They require the following privileges:
CREATE SESSION
SELECT on the shipping tables
- ◆ planner: these users are concerned with coordinating the overall process and, perhaps, providing reports to management. They require the following privileges:
CREATE SESSION
SELECT on the production tables
SELECT on the sales tables
SELECT on the shipping tables
- ◆ DBA: these are the standard privileges granted via the Oracle 6 DBA grant or the Oracle 7 DBA role.
- ◆ data_owner: this is a dummy user ID that owns all the application data tables. This user (and role) requires the following privileges:
CREATE SESSION
CREATE TABLE
CREATE PROCEDURE
CREATE PUBLIC SYNONYM
DROP PUBLIC SYNONYM
CREATE SEQUENCE
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
unlimited tablespace (possibly, if used by the DBA)
- ◆ operator: in this particular company, the nightly update scripts are run by the operators using their own Oracle user IDs. Therefore, these accounts require the following privileges:
CREATE SESSION
SELECT, INSERT, UPDATE, DELETE on the order tables
SELECT, INSERT, UPDATE, DELETE on the production tables
SELECT, INSERT, UPDATE, DELETE on the shipping tables

SELECT, INSERT, UPDATE, DELETE on the sales tables
ALTER DATABASE (startup and shutdown)
ALTER SYSTEM (kill user processes when needed)

Figure 13.2.
Sample data warehouse
privilege scheme.



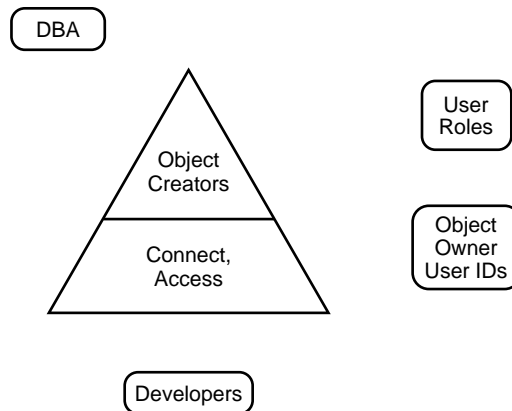
The final configuration is a sample security scheme for a development instance. This is a different environment from either of the two considered previously. Here, data security is not as important as having access to the system privileges required to get the job done. Many development systems go too far in the direction of granting unlimited privileges. Although it may sound good to have people who are not restricted from being productive, you eventually have to implement the controls that you will use in production so that your testing reflects how the system will perform.

Balancing acts like this are always a challenge. The way that I usually like to set up a development instance is to have two classes of developer. The first class contains people who develop software but are not trained or involved with the development of database objects. Perhaps they are good with the front-end tools, but are not sure what the word normalized means. The second class of developer develops new table structures, views, and perhaps even stored procedures. By having two separate sets of roles for these developers, you do not give too many privileges to developers who are not ready for them.

As part of the balancing act, you also create the same object owner account that you will have in production. When the developers have settled down the format of a particular set of tables, they transfer the object creation scripts to someone (developer or DBA, depending on how your organization functions) to be run from the object owner account. All the fine points, such as the creation of public synonyms and the grants of privileges to roles, are added to these scripts.

Finally, it is important to create dummy Oracle user IDs that have roles and privileges similar to those of the actual users of the application. You prove nothing if you do all your testing from the object owner account, which has full access to all the objects that it owns. Because the developer accounts presented in Figure 13.3 have some additional privileges, such as select any, which speed the development process, they are also not valid for testing. You could grant all these roles to the developers and trust that they would set their roles accordingly before testing. However, I do not trust schemes where there is a lot of setup work that needs to be done before conducting tests, and therefore I create these accounts that simulate real world users. The test cases developed then specifically state that they need to be run as a certain user.

Figure 13.3.
Sample development
instance privilege
scheme.



This section has provided you with three samples to consider when you develop your own security scheme. It is important to remember that the security scheme should be tightly tied to the business rules and needs. Therefore, your own individual schemes may be different from those presented in this section. I have tried several means to help develop my security schemes. My current favorite is illustrated in Figure 13.4 (this is also included on the disk as `sec_role.txt`). It has the advantage of including the common roles and grants that I would use, while fitting on a single sheet of paper. Feel free to modify it into something that fits your own style.

Figure 13.4.
Sample role security
checklist.

Oracle Role Security Privilege Checklist							
Application: _____							
Date: _____		Prepared by: _____					
Role Name: _____							
System Privileges:							
<input type="checkbox"/> Create Session	<input type="checkbox"/> Alter Session						
<input type="checkbox"/> Create Cluster	<input type="checkbox"/> Create Procedure						
<input type="checkbox"/> Create Database Link	<input type="checkbox"/> Create Public Synonym						
<input type="checkbox"/> Drop Public Synonym	<input type="checkbox"/> Create Sequence						
<input type="checkbox"/> Create Snapshot	<input type="checkbox"/> Create Synonym						
<input type="checkbox"/> Create Table	<input type="checkbox"/> Create Trigger						
<input type="checkbox"/> Create View	<input type="checkbox"/> Unlimited Tablespace						
<input type="checkbox"/> Execute Any Procedure	<input type="checkbox"/> Select Any Sequence						
<input type="checkbox"/> Backup Any Table	<input type="checkbox"/> Lock Any Table						
<input type="checkbox"/> Select Any Table	<input type="checkbox"/> Insert Any Table						
<input type="checkbox"/> Update Any Table	<input type="checkbox"/> Delete Any Table						
_____	_____						
_____	_____						
Object Privileges:							
Object	S	I	U D E	Object	S	I	U D E
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -
_____	-	-	- - - -	_____	-	-	- - - -

USE OF SCRIPTS TO CAPTURE PRIVILEGE SETS

After a few weeks on the job, most database administrators come to the conclusion that change is eternal. Developers find glitches in their software and need to make changes, new applications are constantly being installed, and every now and then you run into a problem where data is lost and you need to restore it. One of my favorite concepts in Oracle 7, and especially in Oracle 6, is storing all the grant commands that used to set up the instance in a script file on disk. These scripts come

in handy when you have to drop and re-create tables or when you want to compare actual privilege grants to those that you originally created. It also is easier to copy an existing grant line and make modifications to it than it is to type the new command from scratch at the SQL prompt.

It is relatively simple to create these scripts. You can use whatever text editor you prefer on your system to create a file that contains on or more lines just as you would type them at the SQL prompt, terminated by semicolons (;). The last line in this SQL script file should contain a backslash (\). The back-slash is automatically inserted when you use the edit command within SQL*Plus. Here is an example of such a file:

```
grant create session to golfer;  
grant create session to golf_pro;  
grant select,insert on golf_scores to golfer;  
grant select,insert,update,delete on golf_scores to golf_pro;  
\
```

To execute a multiple-line SQL script file such as this, you use either the start command or the at sign (@). The following is an example of executing the previous SQL script:

```
SQL @\odbasg\grants.sql
```

```
Grant succeeded.
```

```
Grant succeeded.
```

```
Grant succeeded.
```

```
Grant succeeded.
```

```
SQL>
```

SUMMARY

By now, you should be comfortable with the idea of Oracle privileges and roles. This chapter covered the basic concept of roles, then went discussed connecting roles, privileges, and users into a security scheme that needs to be developed for applications and instances. Several example security schemes were presented for you to consider when you develop your own schemes. Finally, you explored the concept of storing your privilege grants in operating system script files.

Now that you have learned some of the theory basics of Oracle administration, it is time to move on to a very practical chapter on backup and recovery. The DBA is usually the person given the task of ensuring that the backups are completed and you can recover the data in case of some catastrophe. The next chapter provides you with an understanding of the backup options that Oracle provides (you knew there had to be options for you to study and choose from) and how they can be used in different circumstances.

- 
- The Importance of Backups
 - Overview of Oracle Backup Schemes
 - Archive log or No Archive log
 - Cold Backups
 - Warm Backups
 - Exports
 - Which Scheme to Choose
 - Rotating Backup Schemes
 - Automated Backup Schemes
 - What About Mirrored Disks?

CHAPTER 14

Backup and Recovery

This chapter is one of the more important chapters in this book. Why? Because I have made some really dumb mistakes reconfiguring databases that no one knew about due to the backups that I made before making my mistakes. Because I have lost important data files (and even entire disks with hardware failures) and the users did not lose their life's work thanks to routine backups.

When a disk drive is lost, everyone is nervous. You have the opportunity to make them feel a lot better and look like a hero if you implement a backup strategy that enables you to recover the database and get them back in operation. (Unfortunately, there is no guarantee that they will remember this when it comes time for your next pay raise.)

This chapter has four goals. First, it describes the types of backups that Oracle provides. Then, it provides a discussion that should help you choose the backup schemes that best suit your needs. Next, it discusses each backup scheme in sufficient detail to get you started. Finally, it presents a number of related topics, such as disk mirroring and automating your backups.

In the preceding chapters, you were barraged by a variety of choices that you have to make when setting up your Oracle instance. Perhaps, if you are new to the DBA job, you yearn for a simpler life where you are told what to do. In this chapter, you are given a variety of backups schemes in the Oracle environment. The material is presented so that your decision can be based on a relatively simple set of criteria. Once again, for a product that targets a very wide-ranging audience and variety of computer platforms, this variety is necessary and good.

THE IMPORTANCE OF BACKUPS

Those of you who have worked in the computer support environment for a while probably already understand the importance of backups. You have no doubt seen disk crashes and accidental erasures of data. Let's hope that you have never had to tell the users that the data was not backed up. However, for those who are not experienced computer staff (engineers who are asked to act as a DBA as a collateral assignment, for example) let's look at the importance of backups. I promise to get down off of my soap box after this section and get on to some practical discussions on how to do them.

Almost everyone agrees that the idea of some form of backup is important for computer systems. We admit that things can go wrong with computer hardware or software. It is important to discuss a list of the things that commonly can go wrong to give a true appreciation of the problem. When you get down to the specifics of designing the backup scheme, you design your defense scheme to protect against the common problems. A list of common problems for which you need to have backups includes the following:

- ◆ Complete loss of a disk drive. Mechanical failures can cause the disk drive and all the data on it to be lost without hope of recovery. Some vendors have special utilities that enable them to recover very badly damaged disks, but this could take days in their repair facilities, during which time your system would have to be offline.
- ◆ Accidental (or intentional) erasure of one or more of your key files.
- ◆ Mistakes internal to the application, such as the dropping of a table or a data entry run that enters bad data.
- ◆ Changes made to the configuration files or other tuning efforts that turn out to cause more harm than help.

The sections that follow are designed to give you an understanding of the Oracle backup mechanisms that are available. This will enable you to understand what your options are. Finally, you will go over a method for implementing the system that makes the most sense for different situations.

One final note: Even with the best backup scheme, you sometimes can run into problems. For example, there was a Sun server that had just recently been put into operation where I was working. I assumed database administrative duties for the server. I worked with the system administrator to develop scripts that would enable him to shut down the Oracle database before he performed the nightly cold backup of the data and the operating system. He backed up the system nightly and I reviewed his logs to ensure that things were going according to plans. Then, when a user accidentally corrupted an important table and then dropped it (without thinking of how to recover it), we found out that the backups that had been taken were useless. Most of the system data was stored on raw disk drives. The system administrator had been using the UNIX `tar` utility to back up this data. The version of `tar` that he was using did not back up raw devices (disk drives that are written to directly, bypassing the operating system disk write buffers); it just gave output that looked like it did. The lesson from this is to always test your ability to recover data (perhaps using a test database or a tablespace that you can afford to live without such as the temp tablespace).

OVERVIEW OF ORACLE BACKUP SCHEMES

As mentioned earlier, Oracle presents you with a number of choices from which to build your backup scheme. When I read the chapters in the *Server Administrator Guide*, it always gave me the impression that there were dozens on different backup plans that you may have to implement. They talk about control file backups and partial cold backups and so on. I like simple patterns for my simple mind to comprehend. Therefore, I would say that Oracle provides for two basic types of backups—operating system file backups and object-by-object exports.

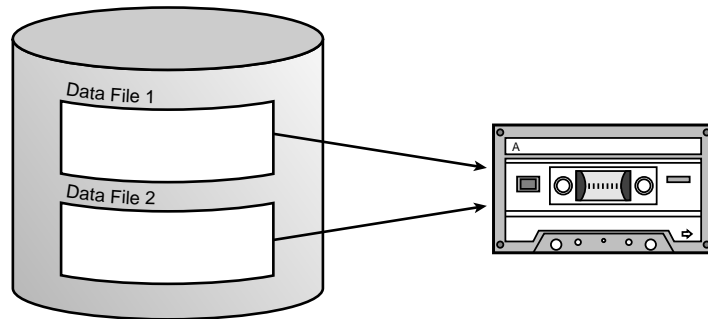
The operating system file backups are the most common ones that I have run across. The basis for this is similar to all other operating system file backups. You use whatever utilities are provided by the operating system (backup, dump, tar, cpio, copy, and so forth) to make a copy of your key Oracle files on a backup medium such as a tape or another disk drive. It's simple and it works. It also enables Oracle to work with a wide variety of operating systems and hardware configurations without having to write extensive Oracle code to control every form of tape drive imaginable. All the operating systems that I have come across have some form of backup utility, so why not use it?

Of course, with databases, you have to be concerned that you get a consistent and complete picture of the data. Complete is a relatively simple concept. You will have problems restoring a database to a replaced disk drive if you have only half of the data files. Consistent is also an important concept. This comes from the fact that backing up a large database (many gigabytes) may take hours. If you have users inserting transactions while you are backing up the data, you would not know whether your updates were made to a data file before or after that particular file was backed up. The easiest way to achieve consistency is to have the database shut down so that it is not receiving any updates from the users. This is what you will see referred to as the cold backups.

This worked well for many of the early, smaller Oracle installations; however, large data warehouses (hundreds of gigabytes) found that there was not enough time available in the day to complete cold backups. You could run the operating system backup commands while the database is operational, but it would be possible to determine whether a transaction was applied to a data file before or after it was backed up and process it accordingly during recovery operations. The warm backup feature takes care of this problem. Basically, Oracle records the start of the backup operation in the log file and when the backup ends. To recover, it starts with the restored data files and applies all log files that have entries from the beginning of the backup to the current time. It is smart enough to know that transactions applied during the backup may or may not be in the restored files. If it finds a transaction that was not applied to the data file before it was backed up, it applies it. This warm backup mechanism is only slightly more difficult to implement and enables you to maintain very high availability for your system.

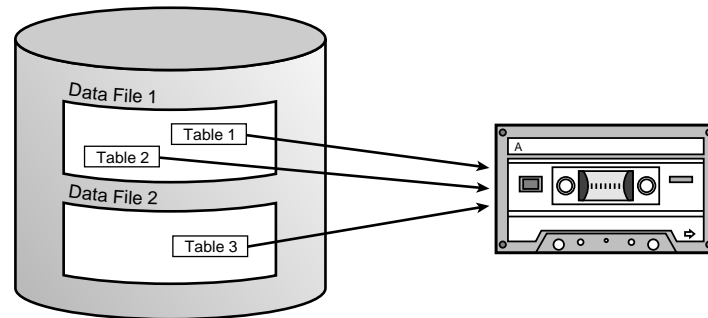
These are the two mechanisms that I have used in the majority of Oracle databases with which I have worked. They have the advantage of being relatively simple and rely upon operating system utilities that should already work with the disk and tape drives that you have installed on your server. They are based on the concept that you take an entire data file at a time and copy it to tape (see Figure 14.1). There is no simple way to restore a particular table that a user accidentally dropped when using cold or warm backups (it can be done, but it is usually so painful and takes such a long time that I have never seen it done). Oracle's solution to the need to store objects such as tables on an object-by-object basis is the Export utility.

Figure 14.1.
Oracle backups.



Oracle's Export and Import utilities are designed to enable you to store and retrieve objects on a table-by-table basis. This file format is the same across all platforms, which makes it a nice tool for copying individual tables between instances that you do not have linked by SQL*Net. I have actually used this as the primary backup mechanism for a remote database that did not have operator tape support. I exported the database to disk, transferred the files up over the network and then backed them up on our local system. The basic concepts of the Export utility are shown in Figure 14.2.

Figure 14.2.
Oracle exports.



You have the same consistency and completeness considerations with Export. If you do not get all the tables, you cannot restore your database to its original state. Oracle version 7 has an option on the Export utility that allows you to specify that a read-consistent backup be taken (`CONSISTENT=Y`). You also lose consistency if there are transactions being applied to tables while you are performing the export. This can be especially troubling when you have multiple tables tied to one another with key fields. If an update is applied to only some of the tables before the export copies the tables to tape, you will be missing links.

Each backup type is discussed in a little more detail in the sections that follow. The key for now is to understand that there are backups that use operating system utilities to make copies of the operating system data files and Oracle exports that

create a new file on disk or tape that contains data that you have specified on an object-by-object basis. Each has advantages and disadvantages and sometimes a mixture of the two approaches works best. The most important considerations when implementing a backup scheme is to guarantee that you have data consistency and that your backup scheme is complete, taking all relevant data associated with a particular Oracle instance. You have the option of cold backups and exports in Oracle version 6. Oracle version 7 supports these two forms of backups plus the warm backup option.

ARCHIVE LOG OR NO ARCHIVE LOG

To archive log or not to archive log—that is the question. The archiving process is Oracle's mechanism to ensure that you have every transaction on file that has occurred since your last backup. In this manner, you take your last backups and apply all the transactions that have been made and you will restore your instance to its configuration before you lost a data file. In effect, you have all the data stored in two locations. First there are the data files. The second location is a combination of the backup tapes and the archive and redo log files. Obviously, if you put your log files on the same disk as your data files and you lose that disk, you lose your data.

Some locations use mirrored disks to provide some data security. Other locations use RAID (redundant arrays of inexpensive disks) technologies that enable you to recover your data from other disk drives in the event that you lose one of the drives in the cluster. I have seen a lot of discussion on the Internet as to whether mirrored disk drives eliminate the need for archive logging. These options provide protection against a disk drive locking up or heads scratching the surface, thereby obliterating the data, but they do not provide protection against a DBA or system administrator accidentally deleting the data files. Yes, you have the power to shoot yourself in the foot. It is not that far-fetched when you consider a large system where you have multiple Oracle instances, each with relatively short names, and you receive four phone calls in the middle of reconfiguring your test instance.

My own personal opinion is that you have to decide whether the risks of losing some data are worth the additional costs for disk/tape space and labor to implement archive logging. That is the fundamental question to deciding whether to use archive logging, whether or not you use RAID or mirrored disks. There are production locations that cannot afford to lose a single transaction. Imagine an order processing system where you are going to have very angry customers if you lose their orders. They will not want to hear about how two disk drives failed at the same time or you typed `ordp` instead of `ordt` for the directory name when you were deleting data files. There are development locations where the developers are just playing with test data that they could easily re-create. In these instances, I have often not used archive logging because it is not a problem if we had to go back to the database as

it was last night. I always recommend considering the pros and cons, documenting them in writing, and then sending your rationale to all parties involved for approval. In the business, this is technically known as covering your posterior.

A few final points about archive logging seem in order here:

- ◆ Archive logging has no meaning if you are using the export/import utilities to back up your database. Only the backup/recovery utilities are smart enough (and have the necessary data) to apply the archive and redo logs to bring the instance up to date.
- ◆ You can archive directly to tape devices on most operating systems. You need to dedicate these devices to Oracle and be sure that you will not exceed the tape capacity before you change the tape. This is often not a problem given the high capacity of modern four- and eight-millimeter tape systems.
- ◆ If you archive to a disk drive, you need to be very careful to ensure that you do not fill up the drive during processing. Oracle has a very bad reaction and can lock up if it cannot write an archive log file due to lack of space. Oracle is programmed to be very adamant about data integrity and not losing data, even in a log file. It's a good idea to implement scripts on systems that could have archive log capacity problems transferring data routinely to tape drives or at least warn operators or the DBA when the archive log directory gets too full.

COLD BACKUPS

Time for a quick review of the term cold backups. The word cold implies that the database is shut down while the backup is occurring. The term backup implies that you are using operating system copy or tape transfer utilities to make a copy of the data. The Oracle discussions in the *Server Administrator Guide* talk about these backups as partial or complete or control file. I like to keep things simple—I start with the concept that I will be performing cold backups and then decide what data files I need to back up and when. More on this later.

Cold backups are the most fundamental form of backup for a database. Exports capture the contents of the data files, but they do not capture their structure. They also do not capture the supporting files, such as control files, initialization files, and log files. On systems where I am performing exports, I usually like to perform a complete, cold backup of the database right after I create the tablespaces, and so forth. I also perform the complete, cold backup after I make any modifications, such as adding or removing data files, control files, or log files. I always recommend the complete backups after such major changes because Oracle has a number of checks and balances built into it (control files capturing the structure, data files capturing

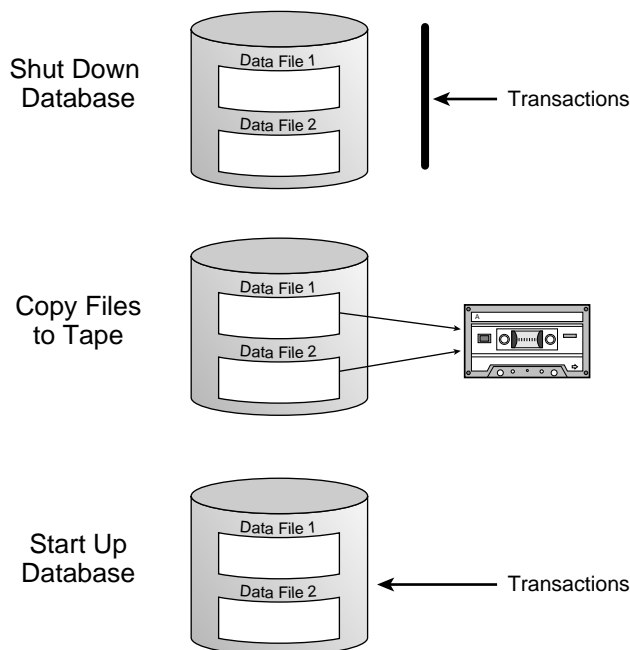
the latest transaction number that has been applied, and so forth). These make it difficult to patch things together, even when it makes sense, given your operational scenario and use of the data.

The cold backups also have the advantage of being an option that is available in both version 6 and version 7. Warm backups enable greater up-time, but they are available only in version 7 of Oracle. Although the number of version 6 databases should be dwindling rapidly, there are still a few out there and this backup means is available to them. An important point to note is that when you perform a cold backup, you are capturing the version-specific information (for example, 6.0.37) in the data files you back up. When you export information, you capture the raw data in a neutral file format that can be imported into a range of Oracle database versions. As a matter of fact, some Oracle upgrades require you to export your data and then reimport it into the new instance.

You should now be comfortable with the concept of what the cold backups are all about and where they fit in the Oracle backup world. You learn how to choose a scheme in a few sections, so don't worry about that yet. Next, let's explore the basics of how you perform a cold backup. Here are the simple steps (see Figure 14.3):

1. Shut down the instance (preferably with a normal shutdown).
2. Copy the appropriate files to tape or another disk drive.
3. Start the instance.

Figure 14.3.
The basics of cold backups.



Actually, the simple description is pretty much the entire description. There are just three points to consider about cold backups. The first deals with the type of shutdown that you perform. I always prefer to do a normal shutdown, because it ensures that all connections and jobs are completed before it shuts down the instance. The shutdown immediate kills off most connections and Oracle may have to do some cleanup once you start again. However, I have found that in most client-server instances and some host-terminal instances, I have to perform a shutdown immediate. The reason is that a single open connection (for example, a user who leaves a PC logged into Oracle when leaving work) can hold up your shutdown immediate and backups. You can build scripts to kill off these users just before your backup, but I have usually found that when I am running scripted backups in the middle of the night (outside of published hours of availability), I am the only user on the system at these times and I know how to stop the backups from running if I need to. Therefore, I use the shutdown immediate because these users are probably not in the middle of update transactions.

The second point to consider about cold backups is the data transfer utility that you will be using. Basically, you have to get comfortable with the data transfer utilities of your host computer system. Having done both database and system administration, I have my own prejudices as to which utilities are better. However, when in just the DBA role, I always try to coordinate my backups with that of the rest of the operating system. If you are uncertain as to whether to use tar, cpio, or dump to back up your Oracle database, have a talk with your friendly system administrators. They should be able to provide you with the scripts you need if you just tell them which files you want backed up on which night of the week. On some systems, they may be the only ones who can access the tape drives and the only help they may require is for you to prepare some scripts to shut down and start up the database properly.

The exact scripts that you use to implement cold backups vary between operating systems (UNIX has Korn and C shells, VMS has DCL, DOS has scripts, and so on). However, the general flavor of a cold backup goes something like this UNIX Korn shell example:

```
# *****
#
# Script:      ftst_cold_backup
# Purpose:    Perform a complete, cold backup to the ftst Oracle
#             database instance using tar to the rmt0 tape drive (8mm)
# System:     Financial test
#
# Revisions:
#
# 7/8/93      Script created (Joe Greene)
#
# *****
```


such as are provided with the DOS 6 backup utilities. Another option is to use the Backup Manager utility provided with Personal Oracle7. I consider using this if I have a tape drive or a second disk drive that serves as the backup target. Otherwise, the limitations of not being able to compress data onto disks makes me tend to back up my data with the database shut down using a utility similar to DOS 6 backup.

The other half of the backup picture contains the recovery techniques that you use to restore data files or the entire database when a problem occurs. Now is the time to consider and try out your recovery strategies. Do not wait to read about recovery until you are trying to recover from a failed disk drive. The recovery strategies are not too difficult. They are based on two simple concepts—what type of backups you are performing and what data has been lost.

The type of backup that you perform will fix a lot in terms of your recovery capabilities. If you are not using archive log mode, any recovery that you perform will be a snapshot of the database as it was when you performed your backup. Any changes since that backup will be lost. If your backups are based on exports, these will also be a snapshot as of the time of backup. You cannot apply archive logs to the imported tables to catch up with the latest updates. Finally, if you are using archive log mode, you have the option of recovering data using the archive and redo log files. You cannot recover completely if you have lost both a data file and one or more of the redo or archive log files.

The second factor determining the type of recovery that you perform is what data is lost. The Server Administrator Guide has a wonderful full-page table that describes your options based on the type of files that are lost. This complex chart can be simplified into the following set of rules:

- ◆ If you lose a control file, but have several other control files, shut down the database and copy one of the other control files to where the missing control file was and you have restored operation.
- ◆ If you lose online redo log files, alter them offline and then drop them. If you were using mirrored online redo log files, you continue functioning as normal. If not, dropping the missing redo log files from the database should get you going again. Create enough redo log files and groups on either a repaired device or a temporary device to keep your database going. Remember to use the `alter system switch logfile` command to switch to another redo log file (or group) when you want to drop the current redo log file or test that you can switch to a newly created redo log file.
- ◆ If you lose archive log files created since your last complete backup, you have not lost any data. You have, however, lost your recovery mechanism and therefore should shut down your instance and perform a complete backup of all the data files.

- ◆ If you lose a data file, you have a number of options to consider. If you were not using archive log mode, you need to restore your last full backup of the database (that means all data files, even the ones that were not damaged). If you were in archive log mode, you can choose to recover the entire database with the database closed to users or recover a single tablespace while the rest of the database is functioning. These two options are described in the next paragraph.
- ◆ If you lose initialization or other supporting files, restore them from the last backup tape. If you are missing a file such as your oratab file, you need to restore this file before new connections can be processed. However, if you lose your init.ora file, you have until your next shutdown to recover it.

As mentioned, you have two options for recovering data files when you are operating the database in archive log mode. One is to perform a recovery of a single tablespace (or multiple tablespaces one at a time) if the tablespace is not critical to other database operations and there is a need to keep the rest of the system operational. For example, if you have multiple sets of users in a database, each with their own tablespaces on separate disks, there is no need to keep the other groups from doing their work while you recover one group's data. However, you should consider taking all the tablespaces that are used in the same application as the tablespace that you are recovering offline while you are doing the backups. Then users won't add data to one tablespace that may need a parallel record entered into the tablespace that you have online. If you have set up roles for each group, you can revoke the create session privilege for the group whose data you are working on to keep them offline. When you are done, you can regrant create session to that group.

In either case, you will need to copy all the backup data files that are needed to replace the damaged ones. Then you have to supply all the archive and redo log files that have been written since the backup files were made. If you have a relatively small transaction volume, these files should already be online and in the correct locations. Oracle will find them and use them. If you have backed the archive log files off to tape, you are prompted to load each of the data files in order and then tell Oracle to apply the transactions from that archive log file. This can be quite tedious in systems with large transaction volumes, but it does have the blessing of preventing you from losing any data.

You may now be wondering how partial and warm backups fit into the recovery scheme. Partial backups are discussed in more detail soon, but conceptually they are a system where you do not back up all the data files every evening. I've had to do this in very large databases where there was not enough time in the day to complete the backups. The key to recovering data in this type of scheme is that you have to save archive log files back to the time where you took your last backup of any of your tablespaces. Therefore, if you have seven tablespaces and back up one tablespace per night, you need to keep seven days of archive log files so that you can recover the

tablespace that was backed up seven days ago. For warm backups, the only trick is that you have to keep archive log files back to the start of the backup that you need to recover from rather than to the end of the backup, as is the case with cold backups.

Earlier I admitted that I tend to prefer to use the command line rather than the menus of SQL*DBA. In the case of recoveries, I break this rule and tend to use the menus. I do not do recoveries very often (perhaps a half dozen times in my life). The menus work well enough and remind me to put in all the information that is needed. Remember, these menus assume that you have restored the data file using your operating system tape transfer utilities and are ready to apply log files to get the data files up to date.

An important concept to remember when working with recovery utilities is that Oracle is designed to be rigid about not losing data or providing inconsistent data. Therefore, it does not enable you to do something that it thinks might be a problem, even though *you* know that it will not be. For example, you lost the disk on which your temporary tablespace is stored. You are not operating in archive log mode, so you want to restore the temporary tablespace (in which there are no permanent objects) from last night's tape and continue operations. Oracle stops you once it detects that you have data files that have inconsistent change numbers applied to them (for example, the temporary tablespace is several hours behind). I have been able to play some crafty games restoring tablespaces when the instance was operating in archive log mode, but you do not want to try even these tricks unless you are absolutely desperate.

This has been a simple presentation on the Oracle recovery process. This was intentional. If you are new to Oracle, you should probably stick to the basics for a while. Most Oracle databases are relatively simple and the basic techniques described previously work perfectly well. On very large instances with extreme up-time pressures, however, I have played some games to recover data more quickly than would be possible using the straightforward methods. Once, when a temporary tablespace data file was lost, I brought the database up part-way, dropped the temporary tablespace, brought the database fully up, and then re-created the temporary tablespace. This saved me several days of restoring a large number of archive log files on a particularly large Oracle instance. However, I would not recommend trying these more complex tricks unless you have a real need and a fair amount of experience.

One final note about recoveries is to test your recovery techniques before you really need them. It will help you sleep better and reduce your antacid consumption if you have seen with your own eyes that your backups really do work. I have run across situations where the system administrator said that his script would back up raw partitions, but when I needed to recover some data, I found out that they didn't. Perhaps the backup script was missing just one little file or had one of those tape

drive settings just a little bit off. Always test recovery of an instance just after you create it. You can even create a special test instance that contains no useful data just to test your recovery techniques. It enables you to work through the process one time before you have a lot of people breathing down your neck because the production instance is down.

WARM BACKUPS

Warm backups are a neat concept for those of us who have to work with really large databases or offer many hours of availability to our users. The simplest backup to execute is the complete, cold backup. You shut everything down and copy all the files associated with a database to tape. If you have used a directory structure similar to the one recommended in the Optimal Flexible Architecture, you do not have to list a large number of files individually; instead, you just specify the directories in which you have located the data for this instance.

However, as the size of Oracle databases grows and user system availability requirements stretch beyond the normal day's shift, warm backups become an attractive alternative. The reason that cold backups were the first to be implemented and still serve as the basis of most system backups today is that they start with the instance in a known state—shut down. There are no transactions in progress. You do not have to worry about whether a particular transaction was applied to a particular table in a particular data file before, after, or during the backup of that data file. You do not have to worry about a transaction being applied to the table after the file containing the table was backed up but before the file that contains the corresponding index is backed up.

However, it is not really complex to teach Oracle how to deal with data received during warm backups. Oracle needs to know when the backup started and when it ended for a particular tablespace or database. Because it cannot know when particular files have been copied to tape, it treats all transactions as though they may or may not have been applied to the database before the files were backed up. For example, if it processes a log file transaction that says to insert a row into a table, but finds that row already there, it assumes that the row was inserted before the data file was transferred to tape and goes on processing the remaining transactions. The key is that it wants the data to be correct when the restoration is complete. It does not complain that the data is not the way it was expecting it before the restoration started.

There are only two new commands to perform warm backups. They can be issued at the SQL command line (SQL*Plus or SQL*DBA) or from the menus in SQL*DBA. You announce that you are starting a warm backup on a particular tablespace by issuing this command:

```
alter tablespace ts_name begin backup;
```

You back up the data to tape or another disk drive using commands that are standard to your operating system such as `backup`, `tar`, `cpio`, or `dump`. Of course, you tell it when you have completed your warm backup so that if it finds data that is not as expected, you are warned. Here is the command to tell Oracle that the warm backup process is completed:

```
alter tablespace ts_name end backup;
```

Let's compare a warm backup script for UNIX with the previous example cold backup script:

```
# *****
#
# Script:      ftst_warm_backup
# Purpose:    Perform a complete, warm backup to the ftst Oracle
#             database instance using tar to the rmt0 tape drive (8mm)
# System:     Financial test
#
# Revisions:
#
# 7/8/93      Script created (Joe Greene)
#
# *****

# First, place all tablespace in warm backup mode using an SQL*DBA
# script

sqldba lmode=y <ftst_warm_start.sql >>$HOME/backup.log

# Next, use tar to back up all of the appropriate data

tar -cvf /dev/rmt0mn /data1/oracle/ftst/* >>$HOME/backup.log
tar -cvf /dev/rmt0m /data2/oracle/ftst/* >>$HOME/backup.log

# Next, bring all tablespaces out of warm backup mode

sqldba lmode=y <ftst_warm_stop.sql >>$HOME/backup.log

# *****
# END OF SCRIPT
# *****
```

The following are a few final points about the warm backup feature of Oracle:

- ◆ You have to use archive log mode for warm backups. You cannot even issue the `begin` and `end backup` commands if you are not in archive log mode because there is no way for Oracle to ensure that it has applied all the proper transactions that have occurred during and since the backup.
- ◆ You need to keep all the archive and redo log files that have been written to since the beginning of the warm backup, not just the end of the backup.
- ◆ There is not a separate set of recovery options for warm backups. You use the same SQL*DBA command `pick` or SQL commands that you use for cold backups.

- ◆ To find out the status of whether a tablespace is in backup mode or not, you can issue this query:

```
select * from v$backup;
```

For example, the following shows that none of your tablespaces are in warm backup mode (you have to translate data file numbers to tablespaces by using a view such as `dba_data_files`):

```
SQL> select * from v$backup;
```

FILE#	STATUS	CHANGE#	TIME
1	NOT ACTIVE	0	01/01/88 00:00:00
2	NOT ACTIVE	0	01/01/88 00:00:00
3	NOT ACTIVE	0	01/01/88 00:00:00
4	NOT ACTIVE	0	01/01/88 00:00:00
5	NOT ACTIVE	0	01/01/88 00:00:00

EXPORTS

The key to remember about exports as a backup mechanism is that they are completely separate from the warm and cold backup utilities. There are several key points to keep in mind when you are considering this option:

- ◆ First, they cannot work with online or archive redo log files to capture data up to the time a disk drive failed. They serve only to capture a snapshot of the tables and objects as they existed when the export was taken.
- ◆ It is the only utility that backs up data on an object-by-object basis. With the other backups, you can recover only down to the tablespace level. The Export utility enables you to pick a particular table that was accidentally dropped without affecting the other objects in that tablespace.
- ◆ The Export utility does not waste time copying blocks from relatively empty tablespaces; however, it does have to perform database queries to gather the results on an object-by-object basis. It can take longer than the warm and cold backups for relatively full tablespaces.
- ◆ The Export utility backs up database objects. It does not back up control files, log files, or initialization files. These need to be taken care of with other backups.
- ◆ The Export utility can work with all parameters specified on the command line or you can have it prompt you for the storage parameters and objects to export. The same options apply to the Import utility.

The basic pattern for export-based backups is that you pick a relatively quiet time for the database or take it in mount exclusive mode (as opposed to open mode). This is necessary because without log files to apply, you can never be sure whether you catch a transaction in progress when you are performing an export of a table. You

can write the exports to disk files or directly to tape drives. If you need to recovery data from your export, you run the Oracle Import utility. This utility stores the data definition language and creates the tables, indexes, and so forth if they are not there (that is, if you have an empty instance).

I do not typically use the Export utility for my mainline backups because it does not have the capability of applying log files to restore the instance up to the time of failure. However, I do use it for special-purpose backups. An example of this is a backup of a particular table and its indexes before I do major data changes to that table (that is, perform a mass update on a particular column). I also use the Export utility when I need to take a table that has run into multiple extents and compress it into a single extent. The basic method for doing this is to export the table and answer yes when prompted to compress extents. You then drop the table and possibly re-create it with your new desired size. You do not have to create the table manually again unless you want to add a little space to accommodate future growth. You then run the Import utility for that table's export file. If you do manually re-create the table, you have to answer yes when prompted as to ignoring whether the object already exists. Be careful to disable any referential integrity constraints (foreign keys) that point to the table being dropped and then re-enable them once you are done.

A final interesting point about exports. You can transfer export files between Oracle instances, even if they are running different operating systems. The internal format of the export file is universal across Oracle systems. You may need to use a host's EBCDIC-to-ASCII character conversion utility when going to or from IBM mainframes, but otherwise regular file copies work just fine. It is easier if you keep the tablespace names and object owners the same between the two systems, but you can change owners and/or tablespace names if you have to when moving files between instances.

This has been the quick tour of using export as a backup tool. I have not used exports as my main backup tool very often. However, it is available for that function. The key to remember is that it only provides a snapshot of the database as of the time of backup. You cannot use logs to re-create the database as it was at the time of failure. However, it can be very useful for supplemental backups because it is capable of transferring data on an object-by-object basis.

WHICH SCHEME TO CHOOSE

It is usually good to have a number of choices available to you. However, when you are new to the Oracle system, it can be a little confusing to choose between all the options that this system provides. This section explores the criteria that to use to

pick a backup scheme. First, it is worthwhile to review some general criteria that would apply to any scheme that you develop:

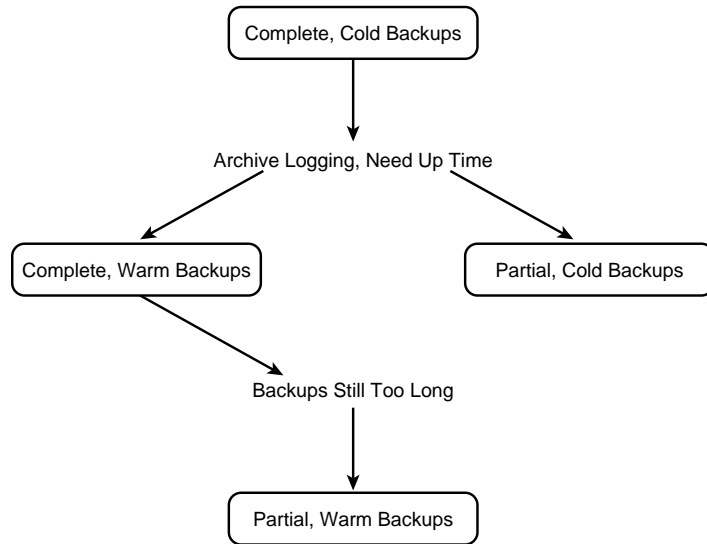
- ◆ The simplest scheme that gets the job done is best, especially for new Oracle DBAs.
- ◆ Consider the way your organization likes to function when designing a backup scheme. Some places prefer to have all jobs submitted by operators and others must run unattended at night.
- ◆ Any scheme you implement must be reliable. To prove reliability, set backup utility options that leave you a detailed log file that you can check to verify that everything has gone as planned.
- ◆ In the end, you need to have agreement from users and management as to what is “good enough” in terms of backups. It is beneficial to document this to all parties—before the first recovery attempt takes much longer than anticipated and you are asked why you did not do things differently.
- ◆ Any backup system that you implement should be tested by a recovery to verify that the system works. I would pick a nonessential tablespace (for example, `temp`) and a test instance to verify your scripts at first.
- ◆ The scheme you choose needs to fit in with the hours of availability and processing for your host computer system.
- ◆ Your backup jobs need to fit in with the operating system backups and any other application processing that may require access to tape resource.

With those preliminaries out of the way, you can now consider the alternatives available under different circumstances. The first factor to consider when devising a backup scheme is whether or not you are in archive log mode. If you are not using log archiving, you are pretty much constrained to performing complete, cold backups of your system. You have acknowledged that you will be able to recover the database only in the event of the loss of a data file up to the point of the last backup. You cannot perform partial backups because if you restore a file, Oracle will detect that this file is out of synch with the other data files and that it cannot correct the problem without archive logs.

If you are using archive logging, you have several options. First, your backup can be either cold or warm. If you have enough hours available for cold backups, always start with the cold backups because they meet the simplicity requirement. You next have the choice between partial and complete backups of the database. Because you have log files that can be used to bring all the tablespaces into synch, you can do a few files each night until the entire database is backed up. You have to retain archive log files for as long it takes to complete one of your backup cycles. The decision between partial cold backups and complete warm backups depends somewhat on personal preferences and on whether you have any relatively calm periods for the warm backups. If you have no down-time available to perform cold backups and you

cannot complete the warm backups in a reasonable amount of time, your only alternative is partial warm backups. This is often the case with very large data warehouses where it can take 10 to 20 hours of run time to transfer all the data to tape. Figure 14.4 shows the general strategy outlined for choosing a backup strategy.

Figure 14.4.
Decision path for
choosing a backup
strategy.



What about export files as part of the database backup strategy? As I mentioned earlier, I tend to use them as supplements to the basic backup strategy rather than as mainstays. I did use them once under Oracle 6 to perform warm backups of a remote database. I structured my exports to be incremental (every table updated since the last export). Then I would get smaller files that I could transfer across the country from the remote computer to our local computer, which had operators who could mount tapes for me. I always prefer the backup utilities to export for recovery purposes. I also use Export more frequently in development instances wherein it is likely that the developers will corrupt or drop a single table accidentally. Because Export enables you to recover a single object at a time, it can be useful in this capacity.

A final issue when choosing a backup strategy is the frequency with which you perform the backups. If you are not using archive log, the frequency of the backup should match the interval over which you feel it is worth the risk of losing data. If you are using archive log, then you have a tradeoff between spending more time performing the backups versus the amount of time it will take to apply all the archive and online redo log files during the recovery. The amount of time to restore log files can be substantial if you have to back them up to tape to free up room on the

archive log disk drives. I worked on one very large instance where it could take several days to retrieve all the log files from tape and apply them to the instance if the failure occurred late in the week.

My basic philosophy has been to back up as frequently as I can based on operational cycles. On small instances, I perform a complete, cold backup every business night. On larger instances, I perform a complete set of backups at least weekly. I back up tablespaces that I cannot operate without, such as system and central data (as opposed to derived data summaries or indexes), even more frequently. Finally, I like to back up small files, such as the control files and initialization files, daily because they take so little time.

Perhaps you thought that this section would contain a magic answer to all your problems, give you some absolute guidance, such as “If your name is George, do a complete, cold backup every other night.” Unfortunately, it is not that easy. However, the criteria in this section will guide you along your way. If it is a small instance and you have the tape capacity and available time to perform a complete, cold backup every night, do it. It is the simplest and safest approach. The other techniques discussed here and the more complex schemes, such as rotating backups, are really designed for more advanced and demanding instances where the downtime is small and volume of data is great.

One part missing from this section is a method to estimate the time it takes to back up a given tablespace so that you can determine which scheme makes sense for you. Unfortunately, given the wide variety of tape devices, data transfer bus configurations, and operating systems that Oracle runs on, the only way you will know is to give it a try. Time the complete, cold backups that you make just after creating the instance and calculating the rate at which this computer/tape subsystem can transfer data. Then pull out this information when you change the backup scheme to figure out how long your scheme will take.

ROTATING BACKUP SCHEMES

If you have a 1G database, a 2.3G tape drive, and a system that is not used except during normal business hours, you probably do not need this section. You can do complete, cold backups every night. This enables you to recover from a problem relatively quickly and all you have to do is leave the right tape in your tape drive every evening. For you in the rest of the world, consider a rotating backup scheme.

You basically need some form of rotating backup scheme when you cannot back up everything that you need in a single night. This could be caused by either a lack of tape capacity or lack of time to complete the tape operations. In either event, the

solution is simple—do a little bit each night until the entire job is completed. This section presents a few considerations and alternatives for scheduling the various backups.

Let’s start with a sample system of rotating backups. It is a large multi-user UNIX computer that has 100G of data in a production Oracle instance and 10G in a development instance. The data is distributed among the tablespaces as shown in Figure 14.5. Also, for sake of this example, the system is capable of backing up 5G per hour and there is a tape subsystem that notifies the operator when new tapes are needed, and so forth. This system operates 14 hours per day, giving the database administrator 10 hours per day of backup time.

Figure 14.5.
Sample rotating
backup scheme.

Production		
Sun	System	1G
	RBS	4G
	TEMP	5G
	BASE_DATA	40G
	BASE_INDEX	20G
	SUMMARY_DATA	20G
	SUMMARY_INDEX	10G
Test		
Mon-Sat	System	1G
Tue	RBS	1G
Tue	TEMP	1G
Mon,Thu	BASE_DATA	3G
Tue,Fri	BASE_INDEX	1G
Wed,Sat	SUMMARY_DATA	2G
Wed,Sat	SUMMARY_INDEX	1G

From this input, we can see that it will take about 20 hours to back up the production instance and 2 hours to back up test. The system administrator will want time to back up the rest of the operating system disks and user files. Let’s assume that this will take 1 hour per night (incrementals) and 5 hours on Sunday (complete). This gives us 9 hours per day, with the exception of Sunday where we get 5 hours.

When constructing a backup schedule, there could be any number of operational concerns that affect how you arrange your backups. Perhaps you download data into the base tables only on Tuesdays and Fridays. You probably will want to back up your data soon after these updates. However, for sake of argument, assume that there are no great operational concerns. Table 14.1 offers a sample schedule.

TABLE 14.1. SAMPLE ROTATING BACKUP SCHEME.

<i>Day</i>	<i>Backups</i>
Monday	Production system and base_data tablespaces
Tuesday	Production system, base_index, rbs, and temp tablespaces
Wednesday	Production system, summary_data, and summary_index tablespaces
Thursday	Production system and base_data tablespaces
Friday	Production system and base_index tablespaces
Saturday	Production system, summary_data, and summary_index tablespaces
Sunday	All development tablespaces

As mentioned earlier, there may be any number of reasons to modify this scheme based upon local needs. However, the following are some of the reasons that I used when I came up with this scheme:

- ♦ You can usually live with much longer delays in restoring service in a development instance than a production instance. Therefore, back up the production instance twice a week so that you have to apply only three or four days' worth of archive logs, at most, to restore a tablespace.
- ♦ The system tablespace is small and is the heart of the Oracle system. You might feel more comfortable backing it up every night. You also might back up the control files and initialization files every night (they are small and do not affect time constraints).
- ♦ I started with the base_data tablespace because its size pretty much guarantees it will take up an entire night's backup.

Here are some final thoughts on constructing rotating backup schemes that may be useful in your situation:

- ♦ This rotating backup logic applies equally well to both warm and cold backups of your Oracle instance.
- ♦ Some tablespaces that do not contain any persistent data (for example, rbs and temp) may be easier to drop and re-create rather than restore from tape. This is especially true in systems that have a high transaction volume and therefore have to apply a large number of log files to recover the tablespace.
- ♦ The needs of a rotating backup scheme may evolve over time as your instance becomes larger or the developers modify the applications. It is a good idea to review your backup logs and plans routinely.

This has been a quick overview of rotating backup schemes. Once again, it did not tell you exactly how to set up your scheme. That depends on a knowledge of your organization, which you have to apply to the general concepts discussed here. You should now be comfortable with the idea of constructing a scheme wherein you do not back up the entire instance every night. Of course, if you have the time and tape capacity, backing up the entire instance in a single session makes recovery much quicker because there are fewer log tapes to apply.

AUTOMATED BACKUP SCHEMES

I've done some work on Oracle systems where the system administrator can either come in early or stay late and perform all the needed backups interactively. There are other locations where standard procedure has all backups done by computer operators who type in the necessary commands and review the output for errors. There are even some instances that are so small that they can easily be backed up over lunch. However, for larger instances, it is usually beneficial to develop scripts that perform all the tasks needed to complete the backups. These scripts can also be useful for those of us who have problems typing in a long series of commands with the phone ringing and people stopping by.

This section introduces the concepts of developing and running automated backup scripts. It also provides an example set of scripts that can be used as a template for your backup scripts. Oracle backups are a combination of operating system commands (for tape operations) and Oracle commands (to shut the instance down or place a tablespaces in backup mode). Therefore, backup scripts usually consist of a series of operating system scripts and SQL scripts.

There are a number of operating system scripting languages. UNIX alone has three common shells—Korn, Bourne, and C. Although the exact syntax may vary between operating systems, the general format of a cold backup operating system shell script is something like this:

- ◆ A call to SQL*DBA to shut the instance down
- ◆ One or more tape commands to back up the data, control, log, and/or initialization files
- ◆ A call to SQL*DBA to start up the instance

Use SQL*DBA to run these scripts because it enables you to run a script with DBA privileges without hard-coding a logon ID and password in a script. Instead, if you are a member of the DBA group on the operating system, you have permission to connect internal, which gives you all the Oracle privileges of the SYS superuser. Because operators are usually trusted users anyway, it is permissible in most places

to give them DBA privileges. The script itself is usually fairly simple, similar to the following:

```
connect internal;  
shutdown immediate;  
exit;
```

A few points about this script. I tend to use `shutdown immediate` to shut down the database because I have observed a few users who do not log off their systems. These connections cause shutdown and really mess things up. Shutdown normal is usually preferable because it guarantees that the database is in a stable condition and there are no transactions in progress. Because many places do not have people working after midnight, this is a fairly safe practice. The startup script substitutes the startup command in place of the shutdown immediate.

You also may need to consider the fact that scripts often run with a slightly different shell than the normal interactive login. For example, in UNIX, the shell that you execute when logged in is set in the password file. However, if you run a job out of the `cron` utility, you execute whichever shell is set in a separate system configuration file. You do not get the benefit of any configuration parameters that you set up in your `.login` or `.profile` files (the equivalent of DOS's `autoexec.bat`). Therefore, you usually want to specify the shell and explicitly set up needed configuration parameters (such as your path and `ORACLE_HOME` variables) in your batch scripts.

Another consideration for batch scripts is that you may want to capture the results of the scripts that you execute from an automated utility such as `cron` for review in the morning. I have found an amazing number of problems that can crop up and prevent your backups from running properly. Some are related to the operating system or unusual conditions within Oracle. Most operating systems enable you to transfer the output of the script to a file for later review. In the Korn shell, you redirect the output (`1>`) and error (`2>`) streams with a command similar to the following:

```
sqldba lmode=y <backup_script.sql 1>$HOME/output.log 2>$HOME/error.log
```

There are a number of tricks that you can play with these files to enhance the output. I have set up batch scripts that scan the log files for text strings that indicate problems and automatically send electronic mail to the DBAs so that they can review it. If there are no errors detected, a successful completion message is sent.

The exact backup commands vary widely. Each operating system has its set of tape transfer commands (`cpio`, `tar`, and `dump` on UNIX, `backup` on VMS, and so forth). You can also buy third-party tape utilities that you may want to use for backup. However, for reference purposes only, here is an example of a UNIX `tar` command that will backup up data files to tape. For this example, assume that you have all

your Oracle data, control, log, and so on files in three directories (/d1/oradata/prod, /d2/oradata/prod, and /d3/oradata/prod). The command would look like this:

```
tar -cvf /dev/rmt0 /d1/oradata/prod /d2/oradata/prod /d3/oradata/prod
```

Your actual commands and directory names may vary. However, you get the general idea that it is not Oracle that is performing the tape transfer; it is your host operating system. If you have any questions about utilities, options, and so forth, see your system administrator.

To start up your Oracle instance, you execute a SQL script similar to the following:

```
connect internal;  
startup;  
exit;
```

An entire book could be written on the subject of setting up these scripts. The exact syntax and games that can be played are determined by the host computer shell languages that are available. Once you understand the basics of what you want done, you can usually talk your system administrator or some of your developers into helping you with the syntax of the shell scripts.

WHAT ABOUT MIRRORED DISKS?

Some folks believe that if they have *mirrored disk drives*, they do not need to do backups. For those of you not familiar with the term, some operating systems and disk drive arrays enable you to designate two disk drives to be exact copies of each other. If one of these disk drives fails, the other takes over the load with no downtime. This greatly increases reliability, but almost doubles your disk costs. There are more complex storage algorithms (RAID) that provide the same functionality with fewer disk drives, but the general concept is the same. If you lose one disk drive, your system continues to operate.

From this brief introduction, you may conclude that you do not need to perform logging or backups if you have these duplicated disks for your data files. I have never seen both mirrored disks in a pair fail at the same time. However, if you accidentally issue an operating system command to remove a data file on the mirrored disks, both copies will be erased. It all comes down to a question of how much risk you are willing to take. I usually prefer mirrored disks, archive logging, and backups for production instances. For development instances, nonmirrored disks and backups without archive logging is usually sufficient. You have to decide what risks you are willing to take. As a minimum, even if you believe mirrored disks are good enough, do complete, cold backups before and after every major structural change to the database (adding or moving files, and so forth).

SUMMARY

This chapter covers a lot of ground on an important topic. You first learned the types of backups that Oracle has available. Next, you explored some criteria that you need to decide which backup scheme is right for you. Finally, you reviewed methods, such as rotating backup schemes, automating backups with scripts and mirrored disks, to consider when devising the scheme for your instance. Backups are an important topic for every DBA to consider and you should routinely review your backup plans to ensure that they still meet your needs. You also should review the output of your backup sessions to ensure that the backups are being completed as you planned. Finally, you really should test your ability to recover the files that you have backed up.

This concludes the section of this book devoted to the basic technical foundation of Oracle. There has been a fair amount of theory and, of course, my personal opinions put out for your consideration. From here on, there are a lot more day-to-day action topics. The next section starts with the process of installing and upgrading the Oracle software on your server. This process is often one of the more difficult tasks that the DBA has to perform and, unfortunately, it is often one of the first tasks that the DBA has to do. The next six chapters provide guidance on this process.

- 
- The Life Cycle of an Oracle Database
 - Choosing Products and the Environment
 - Planning an Oracle Installation
 - Oracle Installations
 - Planning an Oracle Upgrade
 - Oracle Upgrades

P A R T III

Installing and Upgrading the Oracle Software

- 
- Product Selection
 - Planning the Installation
 - Installation
 - When to Upgrade
 - Planning Upgrades
 - Upgrading the Oracle Software

CHAPTER 15

The Life Cycle of an Oracle Database

Congratulations! You have now arrived at Part III of this book, which means that you have completed the basic technical foundation chapters. This chapter is the first of six devoted to something that takes very little time overall in the life cycle of the database, but can cause some of the most severe problems. Many of the tasks in this job are under your control—where and how to add tablespaces, what is the user security scheme. When you make a mistake, you usually figure it out right after you hit the Enter key. You also have the power to fix it yourself. If, however, you receive a bad set of tapes or you are missing a patch that is needed to make Oracle work under your operating system, you're almost always dependent upon Oracle for help. If you jump wildly into upgrading an important production instance without testing the upgrade, you could shut down your business applications and get telephone calls from more managers and vice presidents than you knew your company had.

I'll be honest. I have been through some really painful upgrades and installations. I have seen bad tapes, missing patches, problems that will not be fixed until Oracle's next release, missing client-server middleware, and even an operating system that was so unstable that I could not complete the software installation without several panics (the UNIX word for crash). (*Middleware* is the communications software used in client-server environments.) Now that I have your attention, try not to worry too much. I have had some easy and successful installations also and I believe that things are getting better with the installation and upgrades that I have received. This chapter and the ones that follow are designed to pass on a little advice about how you can plan your installation and perform it so that you have the maximum chance of success.

This chapter starts with a view of the life cycle of an Oracle database from first product procurement through the many upgrades that some databases will go through. Much of this provides support for those of you who have never been involved in the procurement and support of a major software system. For those experienced computer systems and database folks, it provides insight into how Oracle sees the steps in the process and some of Oracle's terminology.

Every database has a slightly different life cycle. There are some agencies that purchase an off-the-shelf application and hardware package knowing that they will not have the budget to perform any upgrades for many years. Others push the capacities of the system and database. They need to keep up with the most current releases of the products. This chapter covers all the aspects of a typical database life

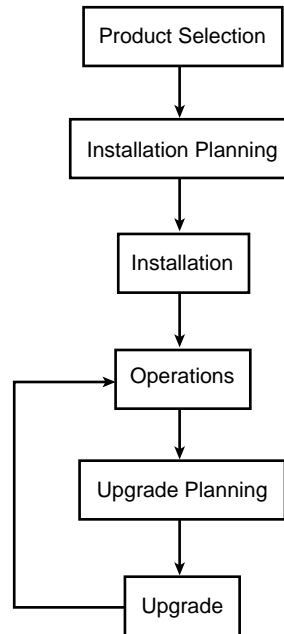
cycle. Not everyone will touch all phases of the process (perhaps the products were procured as corporate standards and you had no input into the selection process). Additionally, the frequency with which you perform some of the steps will vary.

With all of those caveats taken care of, here is the typical life cycle of an Oracle database (see Figure 15.1). There are six distinct phases:

- ◆ *Product selection.* This is where you commit to a particular product suite. This is not a trivial commitment because you are typing your success to the success of one or more vendors. It can be very expensive to change your product suite at a later date, so it pays to get this phase right.
- ◆ *Installation planning.* Don't do an Oracle installation until you have taken the time to plan the process and check the prerequisites thoroughly. You can avoid a fair number of problems just by taking a little time to prepare a sound plan and checklist.
- ◆ *Installation.* Now that your installation plan is done, you can install your Oracle software. You hope that things go exactly as you've planned; however, you should be ready to deal with problems that may arise.
- ◆ *Operations.* This is the goal of the other phases in the life cycle. It is not covered in this chapter, because most of the rest of this book is devoted to it.
- ◆ *Upgrade planning.* Nothing lasts forever. As the demands on computer systems grow and hardware becomes more capable, you will probably be asked to review the latest technology from Oracle to see whether it can help your organization. The key is knowing when to upgrade and implementing a plan that performs the upgrade without interfering with production use.
- ◆ *Upgrade.* Much like the installation phase, the upgrade phase involves loading new software and converting existing databases to run with the new software. A problem during this phase can be very severe because you may corrupt some of the data that your users have been carefully storing in your system. Don't worry, you will never perform an upgrade without performing a good backup, right? Later in this chapter, you look at some of the problems and how to back out of them.

There are chapters devoted to each of these phases. This chapter provides an easy overview before going into some of the details.

Figure 15.1.
Typical Oracle data-
base life cycle.



PRODUCT SELECTION

In addition to playing database administrator, I've done a fair amount of architectural and planning work. There's nothing like the pressure of having to design a system that integrates a number of vendor products and has to work correctly from the start. Upper management is usually intensely sensitive to capital expenditures for computers. You would be amazed at the number of directors and vice presidents with little or no computer background that show up to give advice based on a magazine that they picked up at the newsstand last night.

This is a serious and important task for the computer staff to take on. Many companies are unhappy with the applications that they have had around for a decade or more, but they cannot justify the large expenditures that it takes to convert these architectures and applications. Perhaps in a decade or so, people will be complaining about the difficulty in converting UNIX, Powerbuilder, and Oracle software into what they consider to be modern computer environments.

It is tough in that you are also betting on things beyond your control. A company may look good today, but tomorrow it might be bought out by another company that does not share the same vision of the future. The new management may not be able to keep up with pressures to continuously improve the product line and make delivery schedules. In a sense, you are committing yourself to live with the architecture that looks best to you today, but could cause you problems when things beyond your control happen in the future.

PLANNING THE INSTALLATION

In the product selection process, you commit your organization to a tool set. During installation, you configure the Oracle database. However, as with so many of the other aspects of Oracle, the installation process has a large number of options and considerations. If you decide not to install the procedural option, you will not be able to run PL/SQL routines. If you locate all your control files on a single disk drive and that drive fails, you will not be able to bring the instance up for tablespace-only recoveries. The list goes on, but you get the point.

There are three keys to a successful installation. First, you need to have a good release of the product, which is Oracle's job. Second, you need to have a sound plan that takes into account your needs and the technical requirements to support the installation. Finally, you need to have access to support resources, such as Oracle support, when problems come up during installation. The last two factors are especially difficult for new DBAs performing their first installation of the Oracle software.

The basis of your installation planning efforts is the *Installation and Configuration Guide* that comes with the Oracle software. This book has some useful information about the improvements that have been made with this release of Oracle and some general philosophy for installations. However, the following are especially useful for the planning process:

- ◆ Charts that enable you to size the amount of memory that you will need to run your Oracle configuration
- ◆ Charts that enable you to determine the disk storage requirements for the Oracle software that you will be loading
- ◆ Technical specifications that list the compatibility requirements various Oracle products have with each other and with operating system components

An important companion to the *Installation and Configuration Guide* is the README file delivered with the software (in UNIX, it is located under \$ORACLE_HOME/RDBMS/ADMIN and it is an icon in the Oracle group under Personal Oracle7 for Windows). There are times when the entire installation or upgrade process is changed from that called for in the *Installation and Configuration Guide*. For example, the *Oracle 7.1 Installation and Configuration Guide* for Sequent only mentioned using Oracle's installer to upgrade the database from 7.0 to 7.1. When you looked at the README file, it did not mention installer (which did not work for upgrades). Instead, it had you run a few SQL scripts manually to perform the upgrade. For the majority of upgrade and installation tasks, the *Installation and Configuration Guide* is correct. However, unless you are an extremely lucky person, it pays to check the README to find out what you really need to do.

Now that you have read the combination of the Installation and Configuration Guide and the README file, it is time to plan your installation. The basic calculations of disk space for software and memory for Oracle products are relatively straight-forward, using the charts in the Installation and Configuration Guide. The real challenge comes in planning space and disk allocation for your database files. Here there are considerations that arise for balancing the input/output load between disks, obtaining accurate data storage requirements from the users or developers, and so forth. Finally, you have to address operational considerations, such as whether you will be using archive logging.

As a final step in the planning process, write up a specific checklist/plan for how you will be doing the installation. The Installation and Configuration Guide (or the README file) lists the steps that are needed. However, there are many steps where you have to interpret statements such as “if you are using..., then...,” and you may not like to have to interpret and read while you are concentrating on other things. With the checklist, you also can put checkmarks next to the steps as you complete them to ensure that you have done everything. Finally, the checklist contains the specific details written in about your configuration. For example, the step in the Installation and Configuration Guide may say to specify where you want to put the redo log files. When I write out my checklist, I put the exact locations that I have decided upon on the checklist. Chapter 17 discusses this topic in more detail and shows you several examples that you can review to see whether they fit your style.

INSTALLATION

You’ve read all the books and files. You’ve got a nice, detailed plan in your hand. You’ve scheduled the down-time. You’ve done a complete system backup. Now, you’re ready to try the installation. You know that there can be problems. You have the number for Oracle support and your customer service ID (CSI) number handy.

Actually, I find initial installations to be much easier than upgrades. Although you may have the handicap of having this installation be your first experience with Oracle, you usually have the benefit of having nothing to damage, as is the case when upgrading an existing database. If all is well, you have been given enough time to work out any problems that come up and also learn something in the installation process. DBAs who do not perform their own installations often lack a good feeling for where all the files are that they need.

The installation process is a tough process to develop for a product such as Oracle. The people who build the installation scripts have to deal with a wide range of operating systems, versions of operating systems, memory, and disk configurations. There is no way for Oracle to predict what other products you may have loaded on your system that compete for memory and system resources. You may have tuned your operating system in such a manner that it is incompatible with Oracle. It is

important to keep these factors in mind when you are about to perform an installation. I've never seen a system where I could not install the product; there were just a few that took a while. Therefore, try to avoid installation timelines whereby you are given a half-day to perform the installation.

There are basically three parts to the Oracle installation process for you to consider:

- ◆ Loading the Oracle software from the tape or CD-ROM and setting up ownership for the Oracle operating system account
- ◆ Linking the Oracle software modules together
- ◆ Creating the new database

For PC Oracle, the process is much simpler. You load the CD into the drive, answer a few questions about what to install and where to install it, and the installation proceeds without further incident. The PC product has the benefit of targeting a specific operating system and graphical user interface environment. In any case, it is a much simpler process.

WHEN TO UPGRADE

I like to separate my discussion of when to upgrade from the discussion of how to upgrade. The actual upgrade process itself is a technical exercise for systems and database administrators. The decision process as to when to upgrade is a bit more challenging. This is because the decision on upgrades is a trade-off where you usually do not have all the data about the pros and cons of the upgrade for your particular system.

It may seem odd in an era of information overload that you would lack data to make this decision. You will be keeping up with the industry publications so you will get reviewer comments. The better reviews will have some technical data based upon their testing of the product to guide you in your decision process. You might be able to get some input from people who run systems that are similar to your own (because a data warehouse system has a different set of problems than a transaction processing system). The risk comes in when you install a particular release only to find it has problems with large triggers (which none of the reviewers tested but you use a lot of) or some other key resource to your applications specific to the release of the operating system that you are using.

Weighed against this risk that something could go wrong is the need to keep up with technology. If you did not upgrade to Oracle 7, you would not be using roles that are a great improvement to database administration. If you have a parallel processor system and do not upgrade to 7.1, you will not be able to take advantage of parallel queries that could speed up those nasty queries that bring user complaints. When you read about the plans that Oracle has for the future, you need to consider this new technology to keep up with the demands of your users (and probably additional processors, memory, disk space, and hours in a day to get all of your work done).

The question is not one of whether you need to upgrade your Oracle system. Rather, it tends to be a question of which releases you will install. Version 7.0 of Oracle had 16 releases, including some nonproduction releases. Version 7.1 will have 6, with version 7.2 coming out about one year after 7.1 was first released. There probably are some installations that go for every new release. However, especially if you are in a production business environment that is extremely sensitive to down-time, you may choose to upgrade only every couple of Oracle releases. This is not as bad as it might sound to some of you. Many of these releases are slight maintenance improvements and bug fixes. One release may fix a series of problems with stored procedures or some other database component that you do not use. Perhaps it adds features such as parallel query, but you are using an IBM RS/6000 with only a single CPU.

Balanced against the benefits provided by a particular release is the sensitivity of your organization to a possible problem. If you are implementing your very first client-server system, you may want to hold off for several months after the system is operational to perform any major database upgrades. Users do not care whether it was just a minor bug in the new Oracle software that brought your application down. They just get the impression that the new technology and system is unreliable.

Many of the concerns that have been brought up in this section can be solved with a test Oracle instance. You can load multiple versions of the Oracle software on a computer system (or, of course, on separate computer systems). You can keep your production instance running under the original software while you test out how the new Oracle works with your applications on your computer systems. I have found that as long as you coordinate changes with them (you do not upgrade their test instance to a new version of Oracle two days before a major delivery), developers are much more understanding than end users. Developers are often the ones pushing for new technologies to simplify a portion of their software or speed up their queries.

PLANNING UPGRADES

The upgrade planning process is conceptually similar to planning the initial installation. You still have to look at both the *Installation and Configuration Guide* along with the README file. The one key difference is that the ability to back out of changes made and restore operational capabilities is very important. I have had several upgrades that either I have made mistakes on (especially the first time I tried to upgrade a database) or that had problems with the media, installer, or new database software itself. In each case, I was able to either fix the problem with some help from Oracle tech support or go back to my trusty backup tape to restore operation while I figured out the solution.

Be sure, in the upgrade planning process, to go through the same calculations for disk space and memory required for the Oracle applications that you went through on the initial installation. Oracle has gotten much bigger in terms of disk space and memory usage since I first started working with it. You may have problems if you assume that you can fit the new release into 102M on disk just because that is what the old version consumed. It also is important to verify again the compatibility checklist that you will find in the Installation and Configuration Guide. Newer versions of Oracle may need to have newer versions of the operating system or other support utilities (for example, X Window for the GUI-based Oracle Forms products).

I always prepare my own procedure from the steps listed in the Installation and Configuration Guide. It contains the specifics of my installation (which products I am loading) and also has space for me to put checkmarks next to tasks that I have completed. One of the most important parts of this plan is the back-out procedures in case something goes wrong. I usually show this to management, developers, and anyone else who may be able to give me useful opinions on steps to include. This review also helps you in case of a problem because you have documented what you have done when you are on the telephone explaining your problem to Oracle tech support.

UPGRADING THE ORACLE SOFTWARE

The upgrade process is very similar procedurally to that of the initial installation. This helps because, after a few installations and upgrades, you get a feel for the pattern of things. The following is the basic order of procedure:

1. Load the software from the distribution media onto your computer system and set file permissions so that everyone can access them.
2. Relink the Oracle applications and tools.
3. Shut down the instance under the existing release of the Oracle software, perform a complete backup, and then bring the instance up under the new release of the Oracle software.
4. Run scripts to change some of the internal views and, rarely, the format of the data in existing databases.
5. Bring the database online for testing and operation.

The exact procedure for the upgrade is determined by the versions from which and to which you are moving. There was a substantial migration effort required between versions 6 and 7. The structures were different enough that a major conversion effort was required. However, between versions such as 7.0.16 and 7.1.3, there were only a handful of SQL scripts that needed to be run to update some of the internal Oracle tables and views.

The README file is especially important for upgrades. There have been several releases of Oracle recently that direct you to use the `orainst` utility in the Installation and Configuration Guide. However, the README tells you to run the upgrade scripts manually. This is important because the `orainst` utility does not work properly in these versions. Your only option is to run the scripts manually.

One final point about upgrades. Oracle keeps track of which version of software is to be used on which instance in the `oratab` file. On UNIX-based systems, this file is usually stored under `/etc/oratab`, although some flavors put this file under `/var`. This is a key file in that it tells Oracle where to find a particular instance. It also tells Oracle which version of the software to use when starting up Oracle instances automatically, such as when you reboot the operating system. You need to make sure that this file reflects the new version of Oracle being used for your instance.

SUMMARY

This chapter introduces you to the life cycle of an Oracle database. It begins with an overview of the life cycle. Some DBAs may not get involved with all the parts of this life cycle (for example, your company may procure a tool set including Oracle without asking your opinion). In addition, certain parts of the installation process may require support from other computer support personnel. This is especially true where the DBA is not the system administrator for the host computer.

The following chapters cover each of the phases of this life cycle in more detail so that you have a good understanding of what is happening before you actually have to do it. Much of the background material in the previous chapters will help you understand what is going on during the installation. That material also will help you when you have to make decisions regarding splitting data files for input/output balancing or to support the backup scheme that you have chosen.

- 
- Getting All the Pieces
 - Host-Based and Server-Based Architectures
 - Client-Server Architectures
 - Dealing with Vendors

CHAPTER 16

Choosing Products and the Environment

Given the right set of tools, you have the opportunity to develop and administer a good system that provides the users with what they want and causes you minimal sleepless nights and weekend crises. The wrong set of tools can cause long delays in development efforts, unreliable systems, support nightmares, and cost overruns. This chapter relates some of my experiences and philosophies with regard to picking out the products and setting up the environment. I have had some systems go in smoothly and perform wonderfully. These systems always gave me a good feeling when I watched how they came together and worked. I have also been called in to fix systems that are missing major components. Sometimes the customer had spent a lot of money on tools that they did not need although there were other components missing that were needed to make the system operational.

I have coordinated a number of multimillion-dollar computer acquisitions in different fields. In many of my consulting assignments, I have worked with the staff of relatively large companies. The large dollar value associated with these acquisitions and companies gave the vendors incentives to provide us with above-average service. However, as most aspects of the computer industry become more competitive as computer systems move toward commodity status, I have found a willingness among vendors to provide some support that enables proper evaluation of products and ensures that you get a complete list of components—if you know what to request.

It is unfortunate that, with capital expenditures under such close scrutiny in most companies and organizations and computers being a major portion of these capital expenses, computer people do not receive better training in acquisition and vendor relations. Many of the best acquisition managers seem to have a natural instinct for negotiation and planning combined with a sound understanding of the technology. However, there are a lot of good technical people who cannot negotiate and nontechnical people who try to use their interpersonal skills to assemble complex computer systems. This section points out a few strategies that you can suggest to vendors that will help you get the data you need.

I will try to limit my discussions to those that apply to DBAs participating in the design of database servers. Having spent time piecing together systems, in addition to just their databases, I may tend to take a broader perspective than just the software that you buy from Oracle. This may not be so bad though, because the DBA often winds up having to prove that it is not the database that is too slow when performance problems arise. Therefore, ensuring that the entire system is robust enough to meet your needs may be in your own self interest.

GETTING ALL THE PIECES

There are organizations that, after spending over \$100,000, have a client-server installation that is really nice, fast, and powerful, but unable to function due to one missing component. It was a simple mistake due both to misunderstanding what the

vendors meant and to the salesperson not thinking through the order when laying out prices. The vendor priced SQL*Net TCP/IP for the UNIX server, but did not provide any pricing for the SQL*Net TCP/IP for the PCs that were running Powerbuilder applications. The customer knew that it had to purchase SQL*Net and saw a line item with a nice fat dollar figure on the vendor price quote for SQL*Net. Those who have put in such systems know to ask for the SQL*Net product on both client and server, but without this experience it is easy to overlook this item. The customer staff put a lot of research into comparing products and specifying what they wanted to implement.

Here is a smaller example. I had one customer who had purchased some new disk drives from a third-party vendor. The customer was told that these disks were “fast and wide,” referring to the fast/wide SCSI devices used on UNIX that transfer data at twice the rate of standard SCSI devices. The vendor, who was competing heavily on price, forgot to mention to the customer that he needed a fast/wide SCSI controller in his server to make these drives work. When I asked this question before coming out to do the installation, the customer called the vendor and found out that the additional card was needed. Of course, this came at an additional cost. The customer also needed to upgrade his operating system in order to get a version that supported these new devices.

There are other examples that are just as annoying. For example, when I purchased my first third-party add-on disk drives for some DECstations, I received some really nice drives with some really strange connectors to hook them. It took several months to get the connectors that matched the cables that were used to interface with the computer. This was in spite of the fact that we ordered products that were listed as compatible with our DECstations and that all of the DECstations that I had seen used a different connector than the cables sent with the disk drives.

A final set of horror stories involves software, including Oracle, that had just recently been ported to a particular version of UNIX. The software arrived, but the installers did not work. Once we got the software installed with a lot of manual intervention, we found out that we needed to install several of the optional operating system utilities to make the Oracle software link function correctly. It took several rounds of running the script, getting an error, loading the additional components, and repeating the process to make this work. Once we had the software loaded, we started receiving errors that required patches from the vendor to fix. This same software ran reliably on many other flavors of UNIX and VMS, but it was new to this particular flavor of UNIX and we got to work the bugs out for the vendors.

They are not all horror stories, however. One of the happier installations that I did involved an Oracle development environment that I assembled for a large engineering contract. By that time, I had some reasonable experience with Oracle, first in a host-terminal environment and later in a client-server environment. Having done

actual installations on both the client and the server, I felt comfortable with the ordering process, the product names, and so forth. We had also chosen from fairly standard PCs and UNIX workstations to make up this configuration, so we were comfortable that there would be only a small chance of finding hardware compatibility problems. We had already purchased several of these UNIX workstations, so we were familiar with their configuration alternatives, how the vendor bundled the components, and so forth. Based on this experience, the real challenge was to get the users to define which tools they wanted to have and how large a system they needed. Once that was established, it was easy to obtain a price quote based on a complete parts list and procure and install the system. It was almost too easy.

Another very successful system installation that I worked on did not involve Oracle, but it was just as complex and the methodology can be applied to Oracle environments. We were working with a desktop publishing environment for a large documentation group. We had some familiarity with the publishing package, but not in the UNIX environment. More importantly, we had done a lot of work with engineering UNIX workstations that emphasized power as opposed to graphics. We also had a number of opinions within the documentation group as to whether the next platform should be a UNIX workstation, an IBM-PC-compatible, or a Macintosh platform. Our challenge was to evaluate all the possibilities, make the users feel as though they participated in the process, and have enough procurement justification to satisfy the federal government contracting officer who was in charge of our contract.

Our solution to these problems was to set up an evaluation environment in our facility. We obtained a large amount of vendor loaner equipment for the workstations and publishing software. We had to purchase the PCs, Macintoshes, graphics cards, and monitors, but we purchased these items routinely anyway and felt that they would not be wasted. We configured these workstations for this desktop publishing package and lined them up in a row on desks that were similar to those of the users. We ran a series of timing and performance tests that showed the response times for typical work functions (hard-core numbers for the contracting types) and also enabled the users to come in and perform a series of routine tasks. The numerical results, combined with average scores, were used to select the winners. It was interesting because there were two winners (one for high power users and another for those interested more in office automation functions), but everyone seemed happy. (Well, if not happy, happy with the product although each of the groups was certain that its group should get more of the new equipment than the other groups, who could get by just fine with the old equipment.)

Enough for horror and happy stories. Let's look at some of the issues that I have seen as impediments to selecting and implementing a new computer system successfully. These are interesting to keep in mind when you are assembling your new

configurations, because many of the people may be in situations similar to yours. So, without further ado, here is a list of common “gotchas” to watch out for when assembling Oracle database system configurations:

- ◆ The vendor forgot to mention one piece of middleware that you have to buy from another vendor to make the software you have just purchased work. Perhaps the original vendor was not technically knowledgeable about such things, assumed something existed in your configuration, or was just trying to win on low prices.
- ◆ The vendor forgot to mention the run-time licenses that are required on every machine that runs their product. A classic example is an application developed using Oracle’s SQL*Forms or Oracle Forms. This can turn out to be a substantial cost if you have a large host computer (where license costs are based upon size) or a large number of client workstations.
- ◆ You purchase a product with a large installation base; however, you are only one of the first customers on this particular type of computer. Depending on the quality of the port and the stability of the host computer and application, you may be the laboratory rat who finds all the bugs for the vendor. Certain products that have little interface with the operating system can be easy to port; others can be difficult. Additionally, different applications stress different components within the operating system and therefore may have more problems on operating systems that are weak in a particular area.
- ◆ You build an installation and integration timeline after the vendor technical representative tells you how long it will take him to do it, but then you do it yourself. Unless you are as intimately familiar with the technologies as the tech rep, you will probably take much longer to complete the project.
- ◆ You order a lot of nice new equipment and do not realize until installation day that there is inadequate power, network connectivity, or—my favorite—air conditioning capacity for the new equipment.
- ◆ You forgot to specify some details on the purchase order, such as whether you use Ethernet or token ring for network cards. The vendor will probably choose the wrong one, given the chance.
- ◆ You have a nice local area network that has always been reliable for you when performing your local printing on electronic mail work. However, when you convert your organization to a large client-server system, you start to see slowdowns. The cause of this problem is often that you have connected every workstation, printer, and server in the organization on a single network. The solution is usually bridging traffic into a series of smaller sub-LANs.

- ◆ When you purchased that expensive host-based license for Oracle, you thought that it included everything in the world. However, after the product arrives, you realize that this license does not include SQL*Net for your clients or SQL*Forms for your application development.
- ◆ In order to be cost-competitive, a vendor may propose the minimum solution that will get you started. For example, you have enough memory to load the software and complete the development and initial data download. Then, when you actually put the system into production, you realize that you need more disks to store the data and more memory to tune for performance.

Now that you have seen the potential pitfalls, let's look at a few techniques that can be used to avoid them. If you follow these, you will not avoid every imaginable problem, but you have a better chance of a smooth installation. Here are some of the things you should do when evaluating products and putting together configurations:

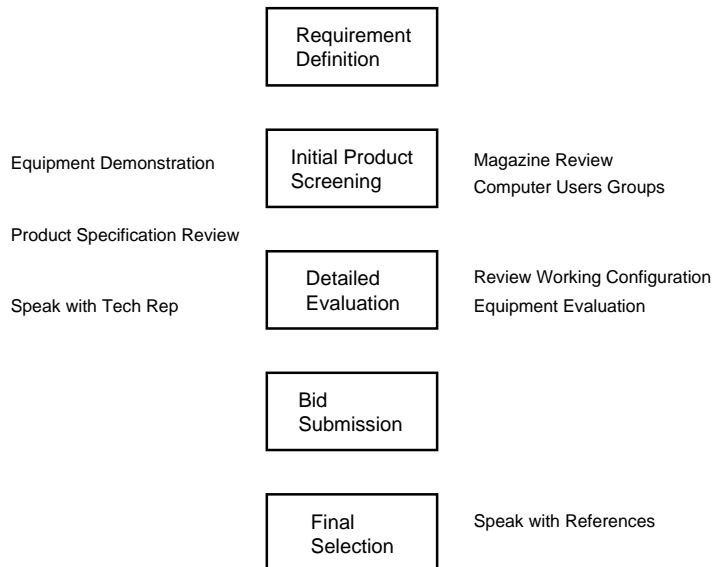
- ◆ Start off by doing as much homework as possible. A good place to start is the many computer industry magazines. Although they do not address your specific environment and needs, they will at least get you familiar with the concepts and vendors. They often do testing that will not prove that the product will work in your environment; but they will usually catch the major flaws that will prove that it would not work.
- ◆ Another good resource is local computer users and management groups. Unless you are in a really unusual situation or industry, the odds are that someone else is going through this process or has already been through it. The people that attend these groups are usually willing to share their experiences. This might help you avoid making the same mistakes that they did.
- ◆ Always ask to see a working configuration for the system. Most computer vendors can make arrangements to load a system with the software that you desire. When you look at this system, look at the detailed configuration (every piece of hardware, software, and so forth that you have to order to replicate this configuration). If the vendor gives you this parts list, it enables you to get familiar with some of that vendor's terminology for such items as disk controllers and also provides a sample for when you prepare your order.
- ◆ Whenever the system is fairly large (that is, you have to get it right the first time), get demo equipment and software from your vendors so that you can set up a model of the system and try things out for yourself. This can catch a number of problems, such as incompatibilities in your network or power and connection problems (for example, cable mismatches).
- ◆ Never, if at all possible, make your first system in a new environment (UNIX, client-server) a mission-critical application that gets attention from

top management. Your knowledge and technique may be perfect from the start, but consider the possibilities of vendor delivery problems, bugs in their software, or equipment failures. Always give yourself time for a learning curve if you are going into new environments (either new hardware or software).

- ◆ Insist on getting the product specification sheets from the vendors for all products, no matter how minor. Look these sheets over closely for incompatibilities between version numbers, product lines, and so forth. Insist on a product specification sheet that matches the products that you are purchasing with all prerequisite products and versions.
- ◆ Always ask how long a particular product version has been on the market and the installation base for the product as a whole and for this version number. Ask for at least an estimate of the number of installations of the product in the type of computer system that you will be using.
- ◆ Go beyond the product sheets to speak to technical representatives or other users who have actually worked with this product in a configuration that is very close to your own. These people can provide you with a wealth of information that salespeople are not likely to know. For example, if you are going to use Oracle in a data warehouse environment, you may wish to go with a multiprocessor configuration, or if you are going to use Oracle Forms on an HP UNIX box, you should avoid certain terminal emulation packages that have problems with keyboard mappings.
- ◆ Beware of the demonstration. Vendors want demonstrations to go smoothly so they work hard on them and practice them several times. Just because the demo system works well is no guarantee that you can actually build applications that have production-sized databases with the products. Always look for someone who is using the product with a similar application in both size and content to be sure that what you are going to try is possible. Oracle has several marketing brochures that show successful (of course) installations in various industries. These can give you a good feel for some other configurations.

The era wherein computer staffs could look to one vendor to supply their computer environment and tell them the direction for the future is, for the most part, over. The pressures to be cost-competitive have caused companies to look very closely at controllable expenses. Therefore, you will probably continue to see pressures to look to new environments and tools for greater productivity and lower costs. With the pressures on vendors for consolidation and acquisition, you also may find yourself switching environments and sales representatives many times. Some of the techniques presented in this section may come in handy for you. Figure 16.1 shows an integrated picture of where these techniques can fit into an acquisition life cycle.

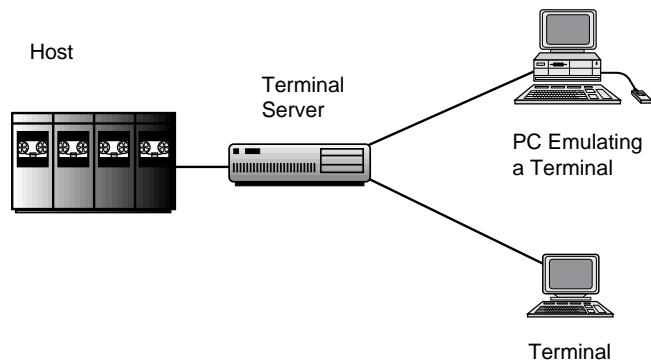
*Figure 16.1.
Sample techniques and
the acquisition cycle.*



HOST-BASED AND SERVER-BASED ARCHITECTURES

Chapter 4 touches briefly on the concept of host-based computer architectures. The basic concept of this architecture is that almost all the thinking is done on the host computer and the terminal is responsible for displaying the characters and graphics that are sent to it by the host. Many companies use personal computers to emulate terminals to communicate with the host computers. However, these PCs do not pick up any of the processing load from the hosts. Figure 16.2 illustrates this environment.

*Figure 16.2.
Typical host-based
computer architecture.*



The great drawback of most host-based computer architectures is that the tools are designed around character-mode terminals. Without a graphical user interface, you cannot provide the easy-to-use windows interfaces that users are beginning to demand. Additionally, most character-based development environments do not provide the high-productivity tools that have come out recently in the workstation world. These environments are the ones that most programmers have learned over the years and therefore they usually require less training.

From a configuration implementation point of view, they have the advantage of centralizing the software and minimizing the complexity of the design of the terminal end of the system. If you use terminals, you usually connect them to a terminal server device, which collects the signals of multiple terminals and routes them to the host computer, usually via a local area network. Sometimes you have a network of terminal controllers that feed directly into special ports on the host computer designed for terminals (as in most IBM mainframe terminal networks that I have run across).

The key to this system from a hardware point of view is having someone who understands the terms and components in the host terminal computer family. The terms used, along with what is considered standard equipment, varies between vendors. A good start is to look at parts lists from similar configurations that exist at the vendor's facility or consult a customer with needs similar to your own. Your only safety device is the specifications that you put out for a quotation. If you do not specify the exact components that you need, the quotes you get back may or may not contain the right components.

Once you have a sound hardware architecture, it's time for the software. Again, you may wonder why you as the DBA have to be concerned about that hardware stuff. The answer, again, is that if there is not enough memory or disk space on the computer, you will be extremely limited as to what you can do to store the data properly or tune your instance. I have found that there are only a few system administrators, tech support staff members, or managers who understand what it takes to make Oracle work well. It is also important to evaluate how well you understand what you are doing. If the requirements are somewhat hazy or you do not have a lot of experience in this new environment, I strongly recommend getting a little extra capacity (disk, memory, and processing) so that you have a safety margin. Computer needs tend to be growing rapidly in most places, so the extra capacity will get used in the not-too-distant future.

When it comes to configuring the Oracle software and supporting tools, nothing beats experience. If you do not have the experience, try to tap into some friends who are doing similar tasks, look to your local Oracle user's group, or bring someone in who understands Oracle configurations to help you build a good one. The Oracle salesman also has access to some technical sales support engineers who can build

configurations for you. The key is to ensure that whoever gives a configuration recommendation can back it up with an example of a similar system (both size and application type) that is out there and working well.

Some specific considerations in the host-terminal environment for Oracle products include the following:

- ◆ If you are running off-the-shelf applications, be sure to verify whether there are any run-time licenses for products such as Oracle Forms required. There may not be, but you need to check because this can be an expensive item, especially in larger configurations.
- ◆ Oracle products often come in bundles. Make sure that you understand exactly what is included in the bundle that the Oracle representative is selling you. This can be especially confusing in large corporations and government agencies where there is a standard bundle and pricing arrangement, which is completed by the centralized purchasing organizations based on the needs that they understand (usually the needs of the first group who requests the product).

The host computer installation methodology is usually more formalized than that found with smaller server architectures. There are usually a series of reviews of the design, requirements, and staffing that will be required to perform the installation. Perhaps most of the installation will be performed by vendor technical service representatives. However, there are some installation considerations that you should consider:

- ◆ Power is often a sensitive issue with larger host computers. They often have options between 220-volt and 110-volt systems. You need to be sure that you have prepared for the type of power system that you will be receiving. The vendor also often requires a specific type of plug to connect the power cable to, so you need to check into this first.
- ◆ You need to plan out whether your host computer will be on regular flooring or on a raised platform. Some vendors have different configuration options between the two or rely on cooling to be obtained through the floor.
- ◆ Always give serious consideration to uninterruptable power supplies (UPS) for host computer systems. They are often more sensitive to power interruptions and take longer to come back up after a power outage. The UPS units also usually provide some form of power filtration and surge suppression that help these computers function better.

Let's wrap up this section with an example of a host-terminal configuration on which I worked. Don't take this as an absolute checklist for your future installations. Oracle and other vendors routinely change their product offerings, names, and bundles to help with their sales efforts. The technical requirements of these

applications continue to grow with each release (more memory, more disk, and so forth). However, it's nice to have something to look at as a template before you start your analytical work.

My example configuration is a VAX system that runs Oracle SQL*Forms and Reportwriter applications that are developed locally. The users all have PCs that run a standard vt100 terminal emulation package from Novell. They communicate with the host computer through a local area network running the TCP/IP protocol suite from Novell. There are usually two developers and five users accessing the separate development and production instances at a given time. The instances are relatively small (a few tens of megabytes) and do not have any intense reports; therefore I was able to get away with a relatively small computer. The following is the configuration that was used for this environment:

- ◆ VAX 4000
- ◆ 64M of RAM
- ◆ 4 450M disk drives (the system had other uses)
- ◆ Oracle RDBMS version 6.0
- ◆ SQL*Forms and Reportwriter development (not run-time) licenses
- ◆ Pro*C (which was not used, but we got anyway under a corporate package deal)
- ◆ SQL*Net 1 TCP/IP (we were experimenting with client-server connections on this system)
- ◆ Procedural option (to run PL/SQL)

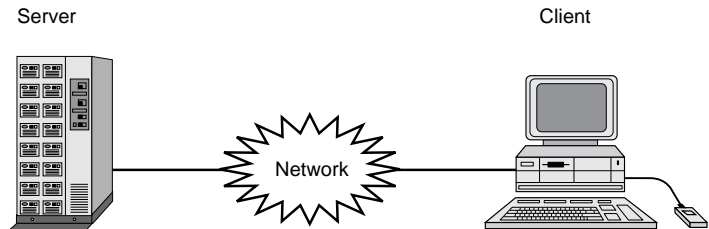
CLIENT-SERVER ARCHITECTURES

A very different architecture from an acquisition point of view is the client-server environment. This environment enables you to give those wonderful GUI-based applications to your users, but it takes some extra setup work. If you have not done it before, you also need to be very careful about the components that you choose to ensure that you get all the pieces needed. This is what you would expect in a younger environment where vendors are scrambling for market shares and standards are evolving. I am not saying that this is an unstable environment. I have worked with several heavy production applications that are client-server-based. However, if you are setting up a new environment or using new tools in an existing environment, you have to be a little cautious when verifying that you have all the products and that they will work together.

A typical client-server architecture is shown in Figure 16.3. In this environment, you need to be concerned about the configurations of both the client computer and the server computer. There are usually two camps of people that you need to bring

together on such designs—the server (UNIX) people and the PC people. This can be a challenge in that many of these people have been allowed to live in their own worlds for some time. The UNIX people understand networking, but have to become comfortable with the networking protocols and middleware (SQL*Net) processes that will be running on their servers. The PC people are used to configuring PCs to run spreadsheets and word processors. They need to become comfortable with the middleware software stacks and be able to tune the windows environment to work well.

Figure 16.3.
Typical client-server architecture.



Here are some of the problems that crop up in client-server installations:

- ♦ Building the proper PC middleware product stack. A product stack is a group of products that are designed to work together to complete a goal. For example, with Powerbuilder on a PC, you may build a stack of the Powerbuilder application, a special dynamic link library module for Powersoft designed to interface with SQL*Net version 1, SQL*Net version 1 TCP/IP, and Novell's TCP/IP software.
- ♦ Network capacity becomes a problem for large installations that have all their PCs connected to a single, nonbridged LAN or that have remote locations with limited transmission capacity (bandwidth) to the remote locations. The load placed on a network that transfers large amounts of data to workstations for further processing is much greater than that of workstations printing to the network printers.
- ♦ Carefully specifying the type of network that you use on purchase orders. This includes the transmission protocol (TCP/IP) and the transmission type (Ethernet or token ring).
- ♦ Ensuring that your workstations are up to the tasks that you are about to place on them. Most client-server applications are designed to provide a lot of functionality to the users. They do not run well on IBM PC/AT computers with 640K of RAM. Most of the installations that I have seen use 486/33-or-better processors with at least 8M of RAM (more for developers).

I actually prefer client-server environments for most database applications. You can keep users off of your server and distribute the processing off of relatively expensive

servers onto relatively inexpensive PCs. If you get all the components, you should be able to assemble an environment that works well. You just need a little time or training to get used to the new world order.

The following are suggestions that should help when you are designing and acquiring a client-server computer environment:

- ◆ The workstations used as servers are relatively inexpensive and can always be used for other purposes. Therefore, you can test out various configurations until you find one that works well in your environment. Many Oracle representatives can bring a demonstration Oracle database on a portable PC to your office and connect it to your network. This enables you to try out several different workstation and software configurations for the client and see how your network responds to the additional load. The workstations can be useful on someone's desktop even if they are deemed too weak or too powerful for the client-server needs.
- ◆ Consider asking a network hardware vendor to evaluate your current network loading to see whether you would benefit from bridging, additional concentrators, and so forth. The vendor has an interest in selling you the additional equipment and you get to see where you stand before you invest in the client-server components.
- ◆ Be careful of the version numbers and protocols for middleware. There are two versions of SQL*Net and many different protocol sets (TCP/IP, IPX, and so forth) that do not work with one another.
- ◆ Try to give the developers a reasonably powerful workstation for development. A workstation that is just adequate for a demonstration of the development tool will probably slow down the development effort and may result in problems of running out of memory, disk space, and so forth.

Now for an example client-server configuration for your reference. Be aware that these products change name and compatibility requirements as time goes on. However, with this caveat aside, this configuration connects Powerbuilder to an Oracle database on an HP UNIX server. This was the initial configuration when there was only a development instance on this server; therefore, there was additional disk and memory capacity reserved for later:

Server Configuration:

- ◆ HP H50 server
- ◆ 4 1G disk drives
- ◆ 64M of memory
- ◆ Oracle RDBMS version 7.0.15 with procedural option
- ◆ SQL*Net version 1 TCP/IP

- ◆ Oracle Forms Development (for commercial applications and local development of host-based applications that mix with the client-server applications)
- ◆ Oracle Reportwriter Development (for commercial applications and local development)

Client Configuration:

- ◆ IBM-compatible 80486 50 MHz computers
- ◆ 12M of RAM (developers), 8M of RAM (end users)
- ◆ MS-Windows 3.1 and DOS 6
- ◆ Powerbuilder development environment (developers) and compiled Powerbuilder applications (end users)
- ◆ The Powerbuilder-to-Oracle dynamic link library (DLL) downloaded from the Powersoft bulletin board (not the one received with the application)
- ◆ Oracle SQL*Net version 1 TCP/IP (on every PC)
- ◆ Novell TCP/IP protocol stack (part of the LAN Workplace environment)

DEALING WITH VENDORS

Those of you who routinely deal with computer salespeople probably have your own bag of tricks and relationships with your vendors. This section is a brief overview of points that may be of help to the rest of you. Many technically oriented computer types may not deal with vendors routinely. Then suddenly, after many years of purchasing a given set of products, upper management is forced to look to new environments. This usually results in a mixture of management and technical staff, who are familiar with the previous environment, having to deal with a large number of salespeople who have the best products but speak a series of terms and acronyms that no one understands.

First, it is important to have a feel for the sales staff of most computer and software vendors. They tend to be sales professionals, not computer experts. Many of them know little more than the buzzwords, specification sheets, and price lists that they have been given. Look for hesitation in their eyes or posture when pressing detailed questions on configurations. There are some who pick up configuration requirements very quickly and can give the correct answers. Others may have been selling network products last week and compilers this week. This is not to mention the problems you may run into with the purchasing department having prejudices toward one vendor, MIS toward another, and the department vice president toward a third.

Another concern is product stability. Always press vendors for an installed base. It is important to get figures both for the product line as a whole and for the particular product that you are buying in the particular environment that you are setting up.

An example is the number of installations of Oracle RDBMS version 7.1.4 under AIX 3.2 on an IBM RS/6000 that will store approximately 10G of data in a data warehouse. You should never be surprised to learn that the product that you were sold is still being beta-tested and therefore will not be available for two months.

Another reality to deal with is the pressures upon you. You are probably still doing your regular job and learning the new environment at the same time. The number of vendors and products is very large. If you are making a decision for a large corporation, you may have access to staff and resources to perform a thorough evaluation of every product on the market. If you are dealing with a smaller environment, you may have to limit the number of products that you evaluate. Your time is money. However, if you do not choose your vendors well, you may be stuck with equipment and software that are below industry standards. Starting with independent reviews of vendors and their market share is usually a good way to narrow the list of products to evaluate. Choosing a development tool based on the fact that this company made a good COBOL compiler that you used for years may not be as wise a criterion.

Another way to get a good response from the vendor sales staff is to be technically prepared. Ask a few hard questions to start the meeting (perhaps even send them a list of minimum questions that you want answered in advance of the meeting) to show them that you are ready to talk seriously. A smart sales representative will usually take this as a sign not to waste half of the meeting with a remedial (and sometimes slanted) discussion of what client-server is. It also takes a lot of the fluffy superlatives out of their speech. You get the answers that you need in a minimum amount of time. You may also suggest to the vendor that he may want to bring a technical support representative to make the discussion more productive. The techies tend to waste less time on glossy brochures and to give you more of the real data that you need.

Another favorite of mine is a demonstration configuration. Always try to get them to load it on one of your machines, if one is available. In that way you can watch what it takes to take one of your configurations and turn it into a working system. If the vendor brings a portable demonstration machine, take the time to look at the system configuration, preferably with a technical representative who can install the product. This is where you can catch a number of the configuration problems that might arise.

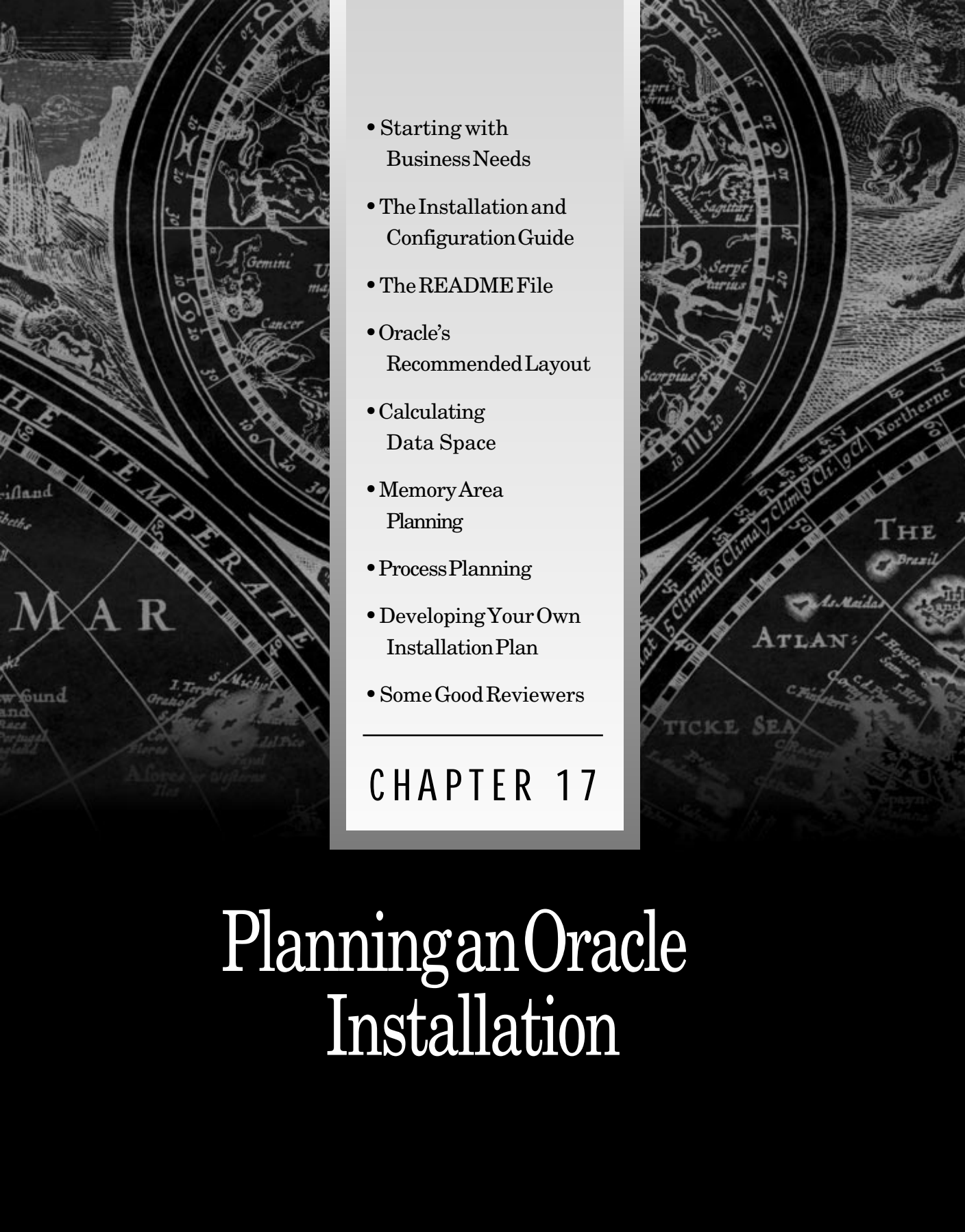
Having the vendor provide a list of references of similar configurations to the one that you are proposing is usually a good idea. Most of these people are willing to take a few minutes of time to discuss their experience with setting up the product. Because they are independent of the vendor, they usually tell you what they like and do not like. They are probably more on your level of experience than the vendor technical representatives and can relate the things that they may have forgotten or problems they might have run into when they installed the system. One key to

remember is that they have to be fairly close in purpose and size to the application that you are proposing. Talking to someone who has a 1M Personal Oracle7 database on their PC has not much bearing on your decision to use Oracle for a 200G data warehouse.

SUMMARY

This chapter covers a number of hints and experiences with building system configurations. Those of you with extensive vendor experience might consider some of the material obvious. If, however, you have not been in an environment where there are a number of computer vendors trying to sell their products to you and speaking in a strange tongue (OSI reference model, open database connection standard, and so forth), you now have a few things to think about before the sales representative gets there. Many DBAs do not have to deal with the product selection process, but if you do, it is to your advantage to get what you need up front.

Once the procurement efforts are over with and the products start to roll in the door, you need to get ready for installation. The next chapter shifts strictly to the Oracle world, and explores planning an Oracle installation. This is a major step and there are often problems with this process. A careful planning approach and a lot of checklists and backups will avoid many problems.

- 
- Starting with Business Needs
 - The Installation and Configuration Guide
 - The README File
 - Oracle's Recommended Layout
 - Calculating Data Space
 - Memory Area Planning
 - Process Planning
 - Developing Your Own Installation Plan
 - Some Good Reviewers
-

CHAPTER 17

Planning an Oracle Installation

By this point in the book, some of you may be getting a little nervous. In the past several chapters, I have described some problems (and even used the words “horror story”) that I have run across loading the Oracle software on various computer systems. You may have forgotten those couple of sentences that I threw in about a number of problem-free installations. You may also have forgotten my telling you how I got around all the problems eventually and got the systems working.

To be honest, I think that the Oracle installation routines have had some real problems in the past. I hope that they will soon get all their installers to the level of the Personal Oracle7 software that I recently installed. This installer asks a few questions on clear, easy-to-follow panels and then completes the installation for you. This chapter is about one of the things that you can do in the meantime to help improve the installation process and get your software installed more quickly: plan the installation. It may seem like a simple concept, but there are (as always) a number of options that you have to consider.

This chapter begins with a rather simple discussion about determining business needs. It may seem obvious, but you may often have some difficulty extracting the details of what the developers and end users want. Without this data, you can only guess at some of the options that you have to have on the Oracle installation utility and later when configuring your instance. This chapter then discusses the documentation that you will receive telling you how to install the Oracle software—the *Installation and Configuration Guide* and the README file. Oracle’s recommended layout for the software and data is presented next, followed by the configuration planning tasks for disk space, memory, and processes. The chapter ends with a discussion of one of my favorite installation approaches (the installation plan) and a discussion of some potentially useful reviewers for your plan.

By the end of this chapter, you should be comfortable with what it takes to install Oracle. You should be able to read through the installation documentation and determine exactly what to do to complete your installation. You will know some of the tricks to capture your plan for review and make the actual execution of the plan a little easier. Finally, you should be comfortable with the options that are available on the basic install.

One final general note about installations. Although the installation procedure on the PC is relatively simple and you do not get much in the way of installation instructions with Personal Oracle7, you should consider some of these concepts when laying out your PC database. Specifically, you should review the file layout considerations and always look for a README file. However, for the most part, you merely have to tell the PC installer where to put things and the installation runs by itself. I sometimes wish it were that easy on some of the larger boxes—but maybe some day.

STARTING WITH BUSINESS NEEDS

Determining the needs of a business is a simple concept. The users and developers tell you what they want and you can implement it. The problems arise when the users describe things in general terms and you are the one left to determine the details. Here are some specific details that you need to have as a DBA before you can construct a sound installation plan:

- ◆ The supporting tools that will be used for application development or operation
- ◆ The rough size of the database and any logical groupings of tables
- ◆ A feel for the daily processing cycle
- ◆ The expected data traffic patterns for the various tables
- ◆ If you are not acting as system administrator, a feel for the disk drives and tape drives that you will be allocated
- ◆ User requirements for data backup, recovery, and reliability

The first requirement that you should be provided with is a list of the tools that will be required to support the database itself. Oracle provides a number of options for development. There are the Forms and Reports generators provided by Oracle and third-party vendors. There are also a number of precompilers that are available as part of the Oracle environment (Pro*C, Pro*COBOL, and so forth). These types of products are relatively simple to integrate. They usually involve merely loading them from the distribution media and linking them with the database as part of the installation process.

Related to supporting tools are two options within the Oracle product structure. The *procedural option* enables you to execute PL/SQL blocks. PL/SQL is the programming language that Oracle enables you to put around the standard SQL statements. It provides such constructs as loops, if-then conditions, variables, and so forth. You may find this option useful even if you do not use PL/SQL programs a lot, because developers using Oracle tools may want to take advantage of stored procedures and other constructs that rely on PL/SQL.

The other option is the distributed option. This enables you to synchronize data automatically between multiple databases connected by a network connection (for example, SQL*Net). There are a few distributed databases out there, but most of the installations that I have seen do not use it. The key to remember about the distributed option is the automatic nature of the database synchronization. You set up the linkages and Oracle works in the background to keep the tables in the two databases synchronized. Most of the installations that I have seen are content with daily program-driven data transfers between the two systems.

If you are using a third-party development tool, you should check its requirements for interfacing with the Oracle database. There may be options required or specific version levels of products that you need to install. The vendor may also require table space or dedicated memory processes to be operational and you need to plan for these items. This process is no more difficult than the one that you use for the Oracle-supplied products. You just need to remember to add this in during your planning process so that you do not have to come back later and reconfigure your system. As always, make sure that the products will work with one another (especially the operating system and Oracle release).

Next on your list of data to collect before planning the instance is a rough estimate for the tables that the users or developers want to create. It is also important to determine whether there are any logical groupings of tables (base data tables, frequently used lookup tables, and so forth). This affects the tablespaces that you design and how you split tables between disk drives to level input/output loads. It also helps you determine the size of supporting tablespaces, such as temp and rbs.

Another important factor when laying out data is an understanding of the expected data traffic patterns for the tables. You may want to locate tables that have heavy usage on different disk drives. If you have a small number of tables that receive extremely high data traffic, you may want to consider disk striping, multiple database writers, and the parallel query option when laying out your plans.

You should also be provided with a feel for the daily processing cycle of the application. This is important when you decide whether to use archive logging or not. For data warehouses wherein data updates occur in a single large batch at night, it is possible to perform cold backups just after the batch upload and not use archive logging. On heavy transaction processing applications, you need to use archive logging to be able to recover all the transactions made during the day. If all the activity is concentrated during specific periods during the day, you may need to consider options such as parallel query and multiple database writer processes.

Because your task is to map user requirements to the Oracle software and available system resources, you need to know what those system resources are. If you are also the system administrator for the server, you can map this out for yourself. If you are not, you need to coordinate with the system administrator to determine the following:

- ♦ The memory that will be available for Oracle and Oracle user processes. In most database servers, you can use all the system memory except that used by the operating system. However, there are other applications that have memory-resident components, so you should check to see whether you are competing with anything else.

- ◆ The tape drives and whether any of them can be dedicated to Oracle uses such as archive logging. Some systems have tape drives that are available only to operators and therefore cannot be used by the DBA for routine tasks.
- ◆ Disk drives. You need to know which ones, how large they are, and what options are available to you for RAID storage, disk striping, disk mirroring, and so forth.

Finally, it is important to get confirmation from the users as to what their requirements are for backup, recovery, and reliability. Specifically, if ultrahigh availability is desired, and the users are willing to pay for it, you can mirror the disk drives so that if one drive fails, you can still access data from the other drive. Some development installations are cost-conscious and some developers are willing to live without disk mirroring or disk space for archive logging in return for lower system costs. The key from the DBA point of view is to document the requirements, get everyone to buy into them, and then construct a database that meets these requirements.

THE INSTALLATION AND CONFIGURATION GUIDE

The basic installation instructions for the Oracle software products are contained in the Installation and Configuration Guide. This document is specific to the operating system and Oracle product release level with which you are working. Never assume that you can follow the same instructions for installing Oracle 7.1.4 on an IBM RS/6000 that you used to install Oracle 7.1.3 on an HP H50. This is almost guaranteed to fail (you have to run some RS/6000-specific configuration scripts before you perform the installation).

The exact contents of this manual vary somewhat over time and with each specific operating system, but the layout is somewhat similar to the following:

- ◆ Overview of the installation process
- ◆ Requirements for loading the various products on the system
- ◆ Issues and restrictions related to this specific implementation of the Oracle products
- ◆ A section on planning for the installation process
- ◆ Pre-installation tasks for the database and other Oracle products
- ◆ Installation of the database and products
- ◆ Post-installation tasks for the Oracle database and other products
- ◆ Overview of the upgrade and migration process

- ♦ Upgrade procedures
- ♦ Information to supplement the standard product documentation that Oracle ships with all operating systems
- ♦ Additional information on upgrading the Oracle tools

The overview section presents a very quick discussion of how to lay out your Oracle database and tools. It presents the standard layout that Oracle recommends for disk files. It also presents some of the issues that you will run across. Finally, it discusses the components of the Oracle server, which may vary as new releases of the product come out, so it is not a waste of time to spend a half hour reading this section.

The requirements section is very important when installing the software for the first time. Sometimes I wish that they shipped this section to you when you are planning out your purchase. Unfortunately, you usually get the Installation and Configuration Guide when the software arrives and everyone is breathing down your neck to have the database ready by that afternoon. This section offers a number of tables that you can copy and fill out to determine the amount of disk and memory space required by the products and the number of users that you are expecting. It also has a list of requirements for each of the products that you should validate before you begin the installation process.

The next section is also important. It provides you with the issues and restrictions that are applicable to your particular release of Oracle and your operating system. The available capacity for such parameters as maximum number of extents in a table vary with the operating system that you are using. This is all internals stuff that the Oracle developers have to decide on when they are performing the port of the Oracle software to your operating system. You do not have a vote in the matter, you merely have to note what these restrictions are and live with them. You get to control how you set up your parameters, up to the limits imposed by the developers for your operating system.

Pre-installation activities involve the system administrator who needs to set up accounts, groups, and directories that you can use during the installation. There are several permission grants that only the system administrator can perform, because Oracle tries to do things with memory areas and file permissions that are typically restricted to the operating system utilities. This is part of the price that has to be paid to get improved performance. These activities are important and should be added to your plan when you prepare it.

The actual installation chapters deal with the tasks that you will perform on the day of installation. There are separate chapters for the RDBMS products and the development tools, CASE products, and so forth. This makes good sense because you

do not have to install the development products in environments such as client-server. Also, you can have installations wherein the development tools may be linked to a database on another server via SQL*Net. I have always found the RDBMS installation to be the more challenging, with the other products usually linking into the environment.

You also have to perform some post-installation tasks on your new software. This includes tasks such as editing the initialization and oratab files and tuning your new instance as needed for your particular environment. You have to set up your user accounts to access the Oracle software (setting environmental variables and placing the Oracle executable directory in the search path). Finally, there is a script that has to be run by the operating system administrator to set some final permissions and turn control over to the Oracle user.

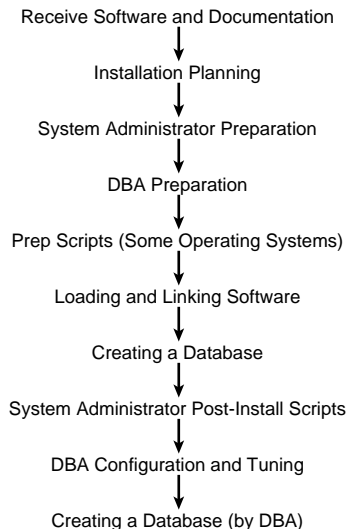
The next chapters that are interesting to DBAs performing new software installations are those that contain additional information on the use of Oracle with this operating system (you get to skip the chapters on upgrades until later in this book). The restrictions chapter is designed to show limits for parameters that are operating system-specific, such as the number of files. These chapters provide the details of how Oracle is implemented under your specific operating system. Contents normally include such items as performance tuning and disk configuration.

Finally, there are a number of appendixes that are contained in this book that provide more detailed information about the installer software, troubleshooting, and multilingual options. I tend to read these sections only when I run into a problem. However, these sections may be of interest to you, especially if you are not using American English, which is the default that I have always used.

Even though the Installation and Configuration Guide books are not fancy, glossy, and bound like most of the Oracle documentation (they are simply printed and bound with binding tape), they go to press well before the software is actually ready for shipping. I have found a number of important changes that are referenced only in the README file. So *always* read the README file after it has been off-loaded from tape. Of course, if you have a CD-ROM drive, you can read the README file before you do the installation—which is even better.

The Installation and Configuration Guide is essential reading both for planning and performing the installation. I have to warn you that this discussion is based on the installation processes that I have seen using versions of Oracle up through 7.1.4. There is always the chance that Oracle will completely revise its installation processes and documentation just to put experienced DBAs on a level playing field with beginners. Figure 17.1 depicts the installation process.

Figure 17.1.
Oracle new installation
process overview.



THE README FILE

The stated purpose of the README file is to show the differences between the actual release of the Oracle RDBMS and related products (SQL*Plus, SQL*DBA, Import, Export, SQL*Loader, and PL/SQL) and the documentation. This documentation has been prepared well in advance of the product release to provide time for the printers and editors to review the material. Although the main features of a given release are fixed early in the development cycle (for example, parallel query for Oracle 7.1), many of the other features wait until fairly late in the product cycle. Many bugs are fixed once the main coding is done on the software. The README file is the only way you know what has been done since the books were sent to press.

The README file is also the best reference for the list of bugs fixed in the product and any final restrictions that were placed on this product since the books were published. The table of contents for this README file usually looks something like the following:

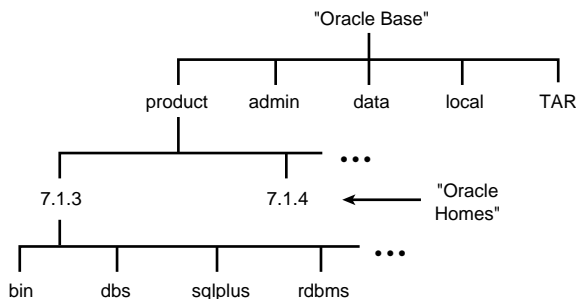
- ♦ A list of documentation available
- ♦ Special migration instructions to convert from version 6 to version 7.x
- ♦ Forward- and backward-compatibility notes
- ♦ Changes made to the Oracle server
- ♦ Upgrading and downgrading notes
- ♦ A list of bug fixes and enhancements in this release
- ♦ Known server restrictions
- ♦ Additional detail on topics that are pertinent to this release and platform

I usually read both the Installation and Configuration Guide and the README file, but tended to believe that the README was always right. Just when I thought I had the pattern figured out, something new popped up. When I went to upgrade the Oracle database from 7.1.3 to 7.1.4 on an IBM RS/6000, I followed the README file, which indicated that I should manually run the upgrade scripts. It did not work and there were problems with the database after I ran the scripts. Fortunately, I followed my own advice and made a complete backup just before the upgrade so I was able to restore quickly. The solution to this problem was to follow the instructions in the Installation and Configuration Guide and use the installer that had not worked for the 7.0 to 7.1.3 upgrades that I had performed just a few months before. (I guess if things were easy, anyone could be a DBA and that would lower the pay scale.)

ORACLE'S RECOMMENDED LAYOUT

When I installed my first Oracle version 6 database, the documentation told me that I needed places to put my software and data. It did not provide any real, clear guidance as to how to arrange my files in a manner that would promote growth in data and software upgrades. Many people installed their systems only to realize that they had to do a lot of rearranging when it was time to grow. To address this problem, the folks at Oracle's consulting arm got together and came up with the OFA. Recall that it is a recommended architecture for locating information on disks. Figure 17.2 illustrates the portions of this architecture that are applicable to Oracle installations.

Figure 17.2.
Oracle's Optimal
Flexible Architecture
(OFA).



With OFA, you allocate a directory that is referred to as the ORACLE HOME (in UNIX, it is stored as the environmental variable ORACLE_HOME). This directory is placed somewhere in the array of disks that matches your other goals. Some versions of the Oracle installation guides recommend placing it under the user home directories. Later versions, however, place it on an applications disk drive that your system administrator allocates for you. I strongly agree with the idea of using a separate applications disk, because newer releases of the software tend to consume

a great deal of disk space and user home data disks are notorious for filling up rapidly. This causes problems when Oracle needs to write an error log file and finds that the disk is full because Charlie insists on keeping 25 versions of all software that he writes.

The next issue when you are planning where to put the software is how much space to allocate for this purpose. There is no magic number. As part of the exercise in the next section, you learn how to figure the amount of disk space for the Oracle software based on which products you are loading. Remember, Oracle makes a wide range of software. It is too difficult to make tapes for each customer based on what that customer orders, so Oracle ships in a few different bundles. This usually means that you get far more software on your distribution tape or CD-ROM than you need for your work. Therefore, you must go through and determine the software packages that you will be using and allocate space for them on the software disk, in the database itself (for support tables and so forth) and in memory.

I always plan on having at least two full releases of the Oracle software on my system when I am performing the sizing exercise. Obviously, I do not want to maintain two versions routinely, and I like to have all databases (production and development) using the same version to make things run more smoothly. However, when Oracle ships out a new version of the database, I always like to run a thorough test against a test or development instance before I try it out on the production databases. I also like to test it against each of the test instances, if there is more than one, because different applications can run into different problems with a new release of supporting software, such as the RDBMS. Of course, if you have far more nerve than I do, you can just let this new, untested software fly on your most important, mission-critical production database. One final note: leave yourself at least a ten- or twenty-percent margin for error when sizing the disk space for your software. The software has grown in size since I have worked with it, so this space helps accommodate your next version. You also may find out that you need additional components (for example, SQL*Net) as your applications and user needs change over time.

The other structure that I tend to use frequently under the `ORACLE_HOME` directory is the admin tree. This is where you will find log and trace files that can tell you what is and has been happening to your database. I also like to put create directories under here to store the data definition language (DDL) SQL scripts that I use to create database objects. I like putting them here because these directories are set up so that only DBAs can access them. In locations where the DBA does not control the DDL, I store these scripts with the application software.

Oracle recommends putting a local subtree for locally developed Oracle software, but I have found that most places have their own directories for applications. If they use version control software such as PVCS, a number of directories are used. There

will be some kind of scheme to control the level of the software modules (production, test, development, and so forth). This local subtree can be useful if you have the responsibility of controlling software builds in addition to being the DBA. Otherwise, put the local applications somewhere else.

The next requirement is some disk space to store your data, log, and other files. In a really small instance, you can put your data on a single disk in a single directory. Generally though, I like to split data files from different databases into different directories, which have names that identify the database. When I restore a test database that has been corrupted, I want to make sure that I am restoring the test data from tape and not the production data. Of course, input/output loading is a function of the disk drive itself, so having multiple directories does not balance this load. Think of it as being a neat housekeeper. The basic concepts for data directories are shown in Figure 17.3.

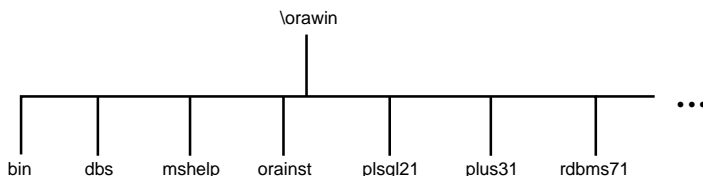
Figure 17.3.
Recommended data
directory layouts.



The exact names of the disks and directories vary between different computer systems. On some operating systems, you may have to specify the disk drive by a letter (for example, Novell o:\oracle) or some other designator. However, the general concepts remain the same. You designate an Oracle Home and some data directories. You lay out the data files depending on the number of disk drives that you have and what the input/output load is that you expect to balance based on application needs.

The exception to these rules is the Personal Oracle7 product for MS Windows (you knew there had to be an exception with Oracle). This is consistent with the general concept of being a good citizen of your environment. In the PC case, most of these systems have a single disk drive and most software applications are stored in a single directory hierarchy off the root directory. Most PC users also are not used to having to search through a complex series of subdirectories the way they do in the UNIX world. Therefore, the Oracle RDBMS software is usually located under the \orawin directory. A separate RDBMS subdirectory is allocated for each version of the Oracle RDBMS (for example, \orawin\rdbms71). This layout is shown in Figure 17.4. If you are used to the PC environment, this probably seems a lot more natural than the OFA.

Figure 17.4.
Personal Oracle7 for
Microsoft Windows
layout.



CALCULATING DATA SPACE

Recall that your first task in the installation process was to go through Chapter 2 of the Installation and Configuration Guide to verify that you have the correct versions of supporting software, operating systems, and so forth. Just after the tables listing the compatibility requirements, you will find a series of tables that enable you to calculate the distribution (Oracle software) space requirements, the database (internal control tables for the particular Oracle tools), and the memory space requirements for the software components that you are loading. This takes the form of a series of tables. Figure 17.5 shows an example table of some selected line items from a typical space requirements table for the UNIX versions of Oracle.

Figure 17.5.
Typical space require-
ments table for Oracle.

SPACE REQUIREMENTS Oracle Server Products

Product	Disk Storage Requirements		Memory Space Requirements		
	Dist (MB)	DB Space (MB)	First User (KB)	#Users	Total KB
Oracle Server	36.60	12.5	6458	489	
SQL*DBA			2589	283	
SQL*Loader			1778	204	
Export			1654	223	
Import			1590	207	
Parallel Query Option	0.08	N/A			
Toolkit II	43.06	N/A			
Common Libraries &	22.10	N/A			
Utilities					
Sub Totals					

DBAs usually photocopy these tables so that they can write on them. Several copies can be useful because you may need to fill in different sheets for each instance. Although you have to load only one version of the Oracle software on a given machine, you have to allocate database space in each application that uses a particular tool. I also find that it is easier to come up with estimates of the number of users on an instance-by-instance (application-by-application) basis for large installations. You will find a number of space calculator tables for the basic server products, the networking products, the development tools, and so on. The goal is to fill out each of these tools as best you can (this is tough when you first start out, so err on the high side whenever possible) and then total the individual sheets to come up with an instance-by-instance total for software space, database space, and memory.

MEMORY AREA PLANNING

A few additional thoughts on memory area planning are in order. The estimates that you obtain from filling out these requirements tables are based on the minimum needs for the tool itself. For example, suppose you come up with an estimate of 7.6M for SQL*Forms 3.0 Runtime for your estimated 10 users. This means that 7.6M ensures that SQL*Forms Runtime will function for you. It does not take into account the fact that if you are running a complex decision support application, your performance will be severely degraded if you do not allocate an additional 10M to sort the large lists of values that the application returns. It also does not take into account the larger log and database buffer space that is needed for intensive transaction processing environments.

Don't worry—there is a chapter coming up on tuning that discusses these subjects. For now, just realize that the numbers that you come up with for memory do not take these factors into account. I always recommend buying a little extra memory. Buying 2G of memory for a small system is a waste of money. However, trying to get a large system functional with 32M is almost impossible. Chances are, if you buy a little extra memory, your applications will grow and new releases of the operating system and Oracle will probably need it soon anyway.

PROCESS PLANNING

Even though it is not really required in the basic Oracle installation procedure, take a little time to plan out the processes that you will be using in your Oracle instances. The installer will create an instance for you that is running the four basic Oracle processes: Process Monitor (PMON), system monitor (SMON), log writer (LGWR), and database writer (DBWR). However, because you have spent so much time planning out your disk and memory usage, take a little time to consider the other processes that you may wish to start up as part of the post-installation activities that you will perform.

Recall the discussion of the Oracle processes in Chapter 9. The following are some of the additional processes that you may want to configure via the initialization files and the rationale for using them:

- ◆ *Multi-threaded server processes.* Consider these on all systems that use SQL*Net and have a relatively large number of users connecting to the system.
- ◆ *Multiple database writer processes.* Consider these if you have write operations that go to multiple disk drives and you have a relatively large write volume on operating systems that do not have asynchronous input/output capabilities.

- ◆ *Checkpoint process.* Use this process only when your log writer process is having trouble keeping up with the volume of checkpoints.
- ◆ *Archiver.* This process is needed if you want to be able to guarantee recovery from the loss of a disk drive (or pair in the case of mirrored disks).
- ◆ *Recoverer.* This process comes into play only when you are implementing a distributed database.
- ◆ *SQL*Net listeners.* If you are planning a client-server installation, you need to start up SQL*Net listeners for each of the versions and protocols of SQL*Net that you are using.
- ◆ *Parallel query processes (version 7.1).* Consider this option when you have databases that execute heavy queries. Typically, this is used in applications such as data warehouses.

DEVELOPING YOUR OWN INSTALLATION PLAN

In one of my previous careers, I drove submarines. In this environment, life was run by a series of written procedures and checklists. (This was a good idea because you really do not want someone to “wing it” when starting up a nuclear reactor.) There even was a written and posted procedure for flushing a toilet. You may think this to be a bit ridiculous, but submariners are quite sensitive about any system that is connected to the sea, especially when you are down deep in the ocean. Anyway, this part of my life taught me to love checklists and plans.

I do not use lists for such routine functions as adding a user to the Oracle database—I use scripts that I have developed to automate this task. These scripts are written to set up default tablespaces and so forth for me. However, because installing or upgrading the database is such a significant evolution and is known to have some problems, I am pretty religious about writing lists and plans. Many of the new DBAs that I have worked with (even the ones that were not in submarines) have commented that they like these checklists and plans. You should consider them for the following reasons:

- ◆ Physically, they are easier to read from than a bound book that won’t lie flat.
- ◆ You have to execute steps from several different sections of the book, in just the right order, to complete the evolution. It is easier to sequence these steps first and execute them from a single list.
- ◆ The written procedure that you have checked off when you complete steps is useful when discussing any problems that come up with Oracle support.
- ◆ If you record start and stop times, along with any notes that you feel are important, on the plan, you can use it for future reference. It comes in

handy when you load another system, create another instance, or just estimate the time it will take to complete this task in the future. In some organizations, it may also come in handy when preparing time sheets.

When I prepare my checklists and plans, I always try to make a list that is “good enough.” I have no desire to write a long-winded work that reproduces every word in the *Installation and Configuration Guide*. I also eliminate steps that are not applicable to this particular installation. They usually relate to steps in the *Installation and Configuration Guide* that begin with something like, “If you are installing the distributed option,...” Finally, I always include the specifics of the directories that I plan to use, tablespace names, sizes, and so forth. I do not like to trust my memory or have to reference a number of sizing calculations when I am actually in the heat of battle. You have to find a style that you are comfortable with (bullets, numbered steps, little detail, lots of detail). Figure 17.6 shows part of an installation plan.

Figure 17.6.
Sample installations.

6. Initial Oracle Instance Configuration Options
 - a. Disk drive configurations:

<code>/usr/oracle</code>	Oracle base application directory
<code>/usr/oracle/product</code>	Oracle home directory
<code>/usr/oracle/product/7013</code>	Install point for this release
<code>/d1/oracle/devel</code>	Development instance directory
<code>/d2/oracle/devel</code>	Development instance directory
<code>/d3/oracle/devel/redo</code>	Storage for online redo logs
<code>/d3/oracle/devel/archive</code>	Storage for archive logs
<code>/d4</code>	Future use
<code>/d5</code>	Future use
7. Preparation Tasks
 - a. HP admin create dba group.
 - b. HP admin create database administrator logins within dba group.
 - c. HP admin create oracle owner account in dba group.
 - d. HP admin create oracle user logins. Note, some thought needs to go in to how the regular user groups will be arranged. There is no need to worry about users who will access only via client-server. This applies to developers and others who will log in to the server.
 - e. HP admin create files in figure 2-3 on page 2-9.
 - f. HP admin alter the Unix kernel parameters as follows:

<code>SHMMAX</code>	<code>0X4000000</code>	{4 million}
<code>SHMMN100</code>		
<code>SHMSEG12</code>		
<code>SEMMNS</code>	<code>128</code>	
<code>SHMMN10</code>		
 - g. Root user create all mount point directories (`/d1`, `/d2`, etc).
 - h. Root user create the directories listed in 6a above with owner oracle and mode 755.
 - i. User oracle set `ORACLE_BASE` environment variable.
 - j. Create the following subdirectories in `ORACLE_BASE`:

-	<code>product</code>
-	<code>admin</code>
-	<code>data</code>
-	<code>local</code>
-	<code>TAR</code>
 - k. Create version 7013 subdirectory under `ORACLE_BASE/product`.
 - l. Set `ORACLE_HOME` environmental variable to `ORACLE_BASE/product/7013`.

The disk contains a quick and dirty installation plan. It works for a specific release of Oracle on an HP UNIX server, so it may not work for you. The Oracle installation procedures and requirements change from release to release and platform to platform. *Please* read the installation instructions that are specific to the version of

Oracle that you are installing and make up your own checklist. You may want to go into more or less detail than the one on the disk, depending on your comfort level with the installation process.

SOME GOOD REVIEWERS

After going through the Installation and Configuration Guide and making a wonderful plan of the installation process that you will be using on your new database, get a second opinion. I usually route the checklists that I prepare to several individuals who can provide me with useful comments. The following people may be able to help you in preparing the ultimate plan:

- ♦ *The operating system administrator.* Look for comments in the area of disk and memory utilization.
- ♦ *The application developers and lead users.* They can provide a good feel for the number of concurrent users, tools that will be used, and so forth.
- ♦ *Your management.* It never hurts to have them look at the plan before you implement it. That way they tend to be less critical when problems arise.
- ♦ *Other DBAs.* It is helpful to have one or more Oracle DBAs in your area to serve as a sounding board, a fresh set of eyes, or the voice of experience.

SUMMARY

Let's hope that you will be permitted to take the time to perform the planning steps described in this chapter and the Installation and Configuration Guide. It is not unheard of to have a tape dumped on your desk in the morning and people expecting to see a working instance by that afternoon. However, you should go through the planning process whenever possible for several good reasons:

- ♦ It is a complex evolution that has been known to have some problems (software bugs, patch tapes needed, bad media, and so forth).
- ♦ The Oracle products have a complex series of interdependencies and require specific versions of operating system components in order to function properly.
- ♦ It may take a lot of time later on to reconfigure disk layouts if you do not implement an architecture that is flexible enough to accommodate future growth.
- ♦ Try using the test instance as a practice run which checks your procedure before you work on the production instance.
- ♦ Having a proper amount of time to plan the installation makes you more comfortable with the process. My earlier comments about problems that I

have run into probably have you a bit concerned. I know I am generally uncomfortable with new installations and upgrades until I have read through the process several times.

This chapter presented a lot of material that you should consider in the planning process. The exact details and steps that are needed are found in the Installation and Configuration Guide. This book also contains the requirements lists and sizing tables that you need to fill out. It is important to work with the Installation and Configuration Guide that is specific to the Oracle and operating system versions with which you will be working. A guide from an older Oracle release or a different operating system will not provide you with the necessary information. It is also important to look at the README file that may contain important steps and notes that did not make it into the Installation and Configuration Guide.

This chapter concluded with a discussion of checklists and plans for installations. They may not solve all of your problems, but they can be very useful.

- 
- Overview
 - Starting with a System Backup
 - The Oracle Installer
 - Installing the Oracle Application Software
 - UNIX Installations
 - Creating a Database with the Installer
 - Dealing with Installation Problems
 - Manually Creating a Database
-

CHAPTER 18

Oracle Installations

You've done your planning. You've read all of the books and README files. Perhaps you've prepared a wonderfully detailed checklist to help you through the installation process. Now it is time to see whether the plans, reference materials, and media that you have received will really work as planned. This chapter covers the steps of the actual Oracle installation, including how to deal with problems that may arise. There also is a small section on how to create a database manually. Although you can probably use the installer to create your instances, this section shows you what is happening behind the scenes and provides a fallback option for you in case the version of the installer that you are working with has problems.

It is not the goal of this section to cover the exact procedure that you will use installing a particular version of the Oracle software on a particular host computer. I have seen a number of changes in the details of the installation process since I have been working with Oracle. It seems reasonable that this process will continue to evolve, becoming more reliable and robust. This section presents the general process that Oracle follows. The documentation supplied with your release can be used to fill in the details.

Before we begin, let's review some of the problems that you may encounter. I honestly believe that the Oracle installation software gets a good testing before the Oracle folks ship it. I have seen a few sets of bad tapes, mostly those that use an older tape format such as QIC. The real problem faced by installation procedure developers is that they must link the application on systems that have been configured by local staff. The system administrator may not have installed all the operating system's optional utilities that Oracle requires (common on many new IBM RS/6000 installations). Software may not be in the directory that would be considered normal for that operating system. Finally, some operating systems (Solaris 2.3, for example) have a large number of patches that need to be applied in a precise order to make the system ready for the Oracle software. It is impossible for the Oracle installation script developers to control all of this from Redwood Shores, California. Therefore, I always try to keep my temper low and attitude positive when I am calling with an installation problem. Of course, if I do not get a return call in a reasonable amount of time, I call the manager on duty—but more of that later.

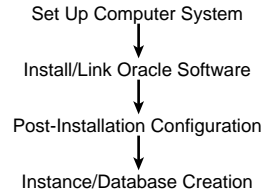
OVERVIEW

What is involved with an Oracle software installation? First, of course, is the planning that was discussed in the last chapter. On the installation day, tasks can be grouped into four categories (see Figure 18.1):

- ♦ Setting up your computer system
- ♦ Installing and linking the Oracle RDBMS and tools software

- ◆ Post-installation turnover of the application by the system administrator to the database administrator
- ◆ Instance creation and configuration

Figure 18.1.
Oracle installation task
categories.



The first step is to prepare your operating system for the new Oracle system. Tasks in this phase include preparing operating system entities such as the Oracle owner account, the DBA group, and the accounts and groups for the users of the system. Because the Oracle RDBMS is more demanding for operating system resources than most applications, you also may have to adjust some of the tuning parameters. This needs to be planned carefully because this requires a system reboot on many operating systems. You also need to create the directory structures for the Oracle applications and data files. Finally, it is always a good idea to check out the available disk space and other supporting utilities (for example, compilation utilities such as `mk` under UNIX that will be used by the Oracle installer) to make sure that everything is as you expect it to be.

The next set of tasks may be performed by the operating system administrator or the database administrator, depending on local preferences and staffing. The process of loading the software from tape usually requires system administrator (root under UNIX) privileges to run any preparatory scripts. Then you load the software from the distribution media—CD-ROM or tape, preferably CD-ROM if you have one. Your basic goal here is to load and link the software for all the products that you will be using (RDBMS, SQL*Net, Oracle Forms, and so forth). You have to be careful on some systems because the operating system utilities, such as the make file on UNIX, may not be installed with the default installation, and it may take some time to get these utilities off the tape and configure them. You also have the option of creating the database at the same time. I usually prefer to have the DBA perform this task because there are a lot of answers that most system administrators cannot answer (they usually accept the defaults).

The third set of tasks is the application turnover by the system administrator to the database administrator. This usually involves having the system administrator run a script supplied by Oracle that sets permissions and other features so that the Oracle user ID can start and stop the Oracle instance and utilities. Typical activities include transferring ownership of the Oracle software from the system administra-

tor user ID (root) to the Oracle user ID and providing access to shared memory. A tricky feature on the UNIX operating system is modifying the permissions on the software for the set UID bit. It changes the shell in which you execute a process to be the same as that of the owner or group that owns the executable file. Oracle uses (and needs) this feature for SQL*Net and several of the other executables. If someone manually starts adjusting the permissions of your ORACLE_HOME/bin directory, you may have problems with this. The root.sh shell script that is used to turn over the application ensures that the software has this bit set.

The final set of tasks deal with instance creation and configuration. As stated earlier, I generally prefer to have a DBA perform this task rather than the system administrator. It is much more work to adjust data files to where you want them, alter rollback segments, and so on after the instance is created. However, if you have to let the system administrator do the task, at least try to specify exactly where you want things to be, their sizes, and so forth.

Each of these topics is discussed in the sections that follow. For now, you should be comfortable with the overall process of a new installation. Note that the installation process for Personal Oracle7 is much simpler, and it automatically creates an instance and database for you. This can be convenient, but you may sometimes want to have more control over sizing and placement of data.

STARTING WITH A SYSTEM BACKUP

Anyone who has read Chapter 14 on backups should remember my problem with the software and that backups saved the day. Many of you may wonder why you would perform a backup before installing the database for the first time. After all, you have no data or software to lose yet, right? Well actually, you do have a lot of software installed in the operating system itself. Even if this is a computer fresh from the factory with no other applications installed yet, it will take some time to reload the operating system and configure it.

This is especially true on modern UNIX systems wherein most of the operating system software is pre-installed at the factory. When you first start up the system, you are guided through configuration by a relatively simple menu system. If, however, you damage components of the operating system (by overwriting configuration files, for example), you have to load from tape or CD. Although not an impossible task, it can consume a lot of time, especially if you are not quite comfortable with the new system and operating system that has been installed for the first time in your organization.

Are the risks of installing Oracle severe and, if so, what are they? Perhaps the biggest risk is the kernel tuning that is often required to bring the operating system

parameters to a level that will support the relatively robust needs of the Oracle database. If you do not have a lot of experience tuning a kernel and working with operating system configuration files, you can adjust your system so that it will not even boot. If you make a mistake in your directory path or save files incorrectly, you can damage configuration files that are needed by the system.

You will probably not run into any problems with your operating system on an Oracle installation. Experienced system administrators may already know the tricks to get around any problems that you do have. However, if you are the nervous type, you will feel a whole lot better if you have that backup tape sitting on your desk when you perform the installation. Most operating system backups are relatively easy to start and you can go off and do other work while the backup is busy writing to your tapes.

THE ORACLE INSTALLER

The basic tool used for the installation of the Oracle software is the Oracle installer. The purpose of the Oracle installer is to make the process of installing or upgrading the Oracle database a relatively simple one—it serves no other purposes. For a given release of Oracle, you will probably see this utility only when you first install or upgrade your database.

A few overview comments are in order about the Oracle installer. First, it is designed for the type of interface that everyone has. For most servers, this means a simple character-based, pull-down menu type system that you see in SQL*DBA and other such packages. The Personal Oracle7 product for Microsoft Windows has a nice Windows-like interface, but that is the only GUI-based version of the installer that I have run across. The trick that you have to remember when using the non-GUI version of this product is that it tries to simulate a button-driven interface in a character environment. You will see several text fields contained in parentheses, one of which will be highlighted. That “button” is what you get when you hit the Enter key. If you wish to select another choice, press the Tab key until that button is highlighted and then press Enter.

For the text-based installer, which is the one most DBAs encounter, there are basically four types of screens. The first is a menu of the actions that you will take, which was shown in Figure 18.2. If your character mapping is correct, you will use the arrow keys to move up or down until your desired action is highlighted and then press Enter (make sure that the Select button is highlighted at the bottom of the screen). The second type of screen is a message asking you to confirm something or select between several choices. It consists of some text and a series of buttons similar to Yes, No, and so on. The third type of screen asks you for a short answer. For

example, the installer will ask you for the `ORACLE_HOME` and `ORACLE_BASE` directories when loading software. In these screens, you type your answer and press Enter. Finally, there is the available products screen. This is perhaps my least favorite feature of the Oracle installer because it is a little confusing. When you are upgrading the database objects themselves, you are supposed to select RDBMS from this menu of available software and it runs a series of SQL scripts to update the internal database views and so forth. With the problems that have existed with the installer in the past, it's confusing to see a screen talking about installing software when you have already installed the software and now want to upgrade the database itself.

How do you get the installer loaded so that you can use it? For most tape-based installations, you need to use a series of tape copy commands (specified in your Installation and Configuration Guide) to transfer some or all of the Oracle software from tape to disk. For CD-ROM installations, you can access the Oracle installer from the CD-ROM to transfer data from the CD to the disks that will eventually hold them. Perhaps I am spoiled, but in my humble opinion CD-ROM is definitely the way to go if you can afford to put a player on your system. Not only can you access the installer directly, but CDs do not become demagnetized or jam or any of the other problems that you can run across with tapes. Their large volume (over half a gigabyte) means that you can store a lot of software on a single disk that fits very nicely in your drawer.

Now for a few tips on dealing with the installer:

- ♦ The first thing you need to set up before trying an installation is your terminal emulation (assuming, of course, that you are not using a real terminal). I usually pick a simple terminal that is supported by my PC terminal emulation software, my operating system, and the Oracle utilities. I have usually found that DEC vt100 terminal emulation works best.
- ♦ Ensure that you set up the Oracle Terminal operating system variable/logical in addition to the host terminal variable. Yes, they are separate and you need to deal with that. In UNIX, for example, the operating system uses a variable called `TERM` to determine how to respond to input; Oracle uses `ORACLE_TERM`.
- ♦ Once you get your terminal emulation set up, you need to become familiar with the keys that Oracle requires to access the menu: Scroll, Tab, and so forth. This is a mixture of understanding the keys that Oracle is expecting (see your Installation and Configuration Guide for details) and where those keys are mapped on your terminal emulation software (see its documentation). You do not really need many keys, and keys such as Tab and the

arrow keys map to the keys you would expect on terminal emulations. The only trick is the key to activate the pull-down menu. The 0 key on the numeric keypad usually does this under vt100 emulation, but you need to check your local setup to access these features.

Before you leave this section, you need to learn about bugs with the Oracle installation process and the Oracle installer. Some computer folks love to rip their vendor's problems and get really upset when something that they have paid so much money for does not work perfectly. My experience with Oracle ranging from 6.0 to 7.1 shows that I have always been able to get the database working. I have also been able to work around most of the bugs in the application software, and the big problems were usually fixed by patches or a new release fairly quickly. However, I have had problems with the installer or the software media in about half the cases.

I am not telling you this to ping on Oracle. I realize that it must be tough to design software that will work on a large range of operating systems that have been configured in a myriad of ways. Oracle has little control over where the system administrator puts things, how configuration files have been altered for other applications, and so forth. You need to give yourself time to work through these problems and have support available to you when it is needed. Although most of the problems that I have run into dealt with the upgrade portion of the installer not working, I have run across releases that needed special patches applied or that Oracle tech support had to go through manually linking the modules with me. Enough said—leave yourself several days to complete the install.

By now, you have had the time to review the manual thoroughly and have made up a little checklist (or photocopied and marked up pages from the installation guide). You also have read the README file and done all the preparation steps listed in the manual (tuned the kernel parameters, set up the accounts and groups, and so forth). The day has come. There is no turning back. You are about to install Oracle.

INSTALLING THE ORACLE APPLICATION SOFTWARE

The exact process used to install the Oracle application software depends on your operating system and the installation devices that you have. As stated earlier, CD is a very nice way to do the installation, but magnetic tape gets the job done. Your first step is to get the Oracle installer running. On some tape-based systems, this usually means copying the core components of the Oracle installer onto disk so that you can use the installer to remove the rest of the applications from tape. Oracle picks a tape transfer utility that is common for the host operating system (backup

save sets for VMS and `cpio` for UNIX). For CD-based installations, you can usually run the installer directly from the CD and it takes care of everything for you.

You hope everything goes smoothly and you do not need to know what is going on behind the scenes. My experience shows that problems come up from time to time and that you need to be aware of what is happening so that you can react properly. First, very few users want to load the entire Oracle software suite. It consumes a large amount of disk space, and it is expensive to license products that you will not be using. Therefore, Oracle ships the component software items separately and uses the appropriate tools on the host computer system (for example, `mk` on UNIX) to assemble its executable files with proper linkages between products.

This is where the first problems often arise. Many of the new UNIX systems are shipped with the operating system software pre-installed. Just as Oracle does not want to take up excessive amounts of disk space, the operating system vendors do not install all the products that you are licensed for when they pre-install the software. They ship the remaining software on tape or CD, from which you can install the other needed components. You may therefore find that the components that Oracle needs to assemble its executables are not installed on your system. In the case of some operating systems such as AIX, you may find that there is an entire hierarchy of components within the software development hierarchy that you have to install in order to enable the Oracle installer to do its job. Coordinate this with the system administrator before the date of installation.

The next set of problems may come up when you actually log in to perform the installation. One of the most common problems is that the system administration account (UNIX's root) and the Oracle owner account are not set up with all the environmental parameters that are needed. One common example occurs when the software linking utilities are installed but they are not in the path of the system administrator. Many systems do not assume that the system administration account will be used for a large amount of software development. Other annoying things that may hold you up are the terminal and Oracle terminal environmental parameters. One of the keys is that when you read the error messages, do not assume that it is an installer problem. If it says something is missing that you know is really there, check the search path and environmental parameters of your user ID to ensure that these are not the problem.

Another common installation problem is applying patches. In spite of extensive testing, additional problems are discovered when software is sent out into the field. By this time, the production organization has made many tapes and CDs to support shipments to the large volumes of users that buy the Oracle products. The solution to this is to ship patch tapes that have to be added to the base software and then

relink the appropriate executable files. This is fine for those folks who are comfortable compiling software applications on their host computer system; however, it can be a little confusing for system administrators and DBAs who are new to the platform or have not done an extensive amount of development. Oracle sometimes provides extra challenges. Once I received a patch where the README file was a UNIX make file (no instructions, just the script that you would execute to apply the patch). Anyway, if you have to apply patches, see whether you can line up some support from someone comfortable with developing software on your host computer system.

There is an entire book written on the subject of installations (the Installation and Configuration Guide—see Chapter 17 for more information). Installations vary between operating systems and releases of the Oracle software. One final suggestion is that you make sure that you still have enough space to install Oracle (and provide a staging area if you are installing from tape). It is quite possible that the amount of disk space that you currently have available has changed from when you started your planning efforts.

UNIX INSTALLATIONS

I have not installed Oracle on all possible platforms. However, on many of the multi-user computer systems that I have run across (especially the older versions of Oracle on UNIX-based systems), the installation is performed by the system administrator (in UNIX, root). The problem is that the tape transfer utilities usually give ownership of the files loaded to the account performing the load. This does not work well for Oracle, which relies on a special Oracle owner account (usually called oracle) to perform most of the administrative tasks. It helps out because many system administrators do not want to give full privileges to the database group. Therefore, some way is needed to transfer ownership of the files loaded to the DBA.

Another item to consider is that Oracle often requires special privileges that are restricted from most other users on the system. For example, many operating systems restrict users from using shared memory areas. Oracle uses these features to improve performance of its software and you cannot get around it. Some mechanism is needed for the system administrator to grant these special privileges to the oracle owner account so that the system will perform as designed.

The tool used to perform this on UNIX systems is the root.sh shell script. After the software is loaded, the system administrator runs this script to set up all the special file access privileges and transfer ownership of the system to the Oracle user ID. This is a step that is sometimes forgotten after you have gone through a long

installation and you are finally elated to have all the patches applied and the software linked correctly. If you have the software installed and root can start Oracle, but no one else can work with it, it is a sign that perhaps you did not run this script or that there is a problem with it.

CREATING A DATABASE WITH THE INSTALLER

So far, you have explored the tape transfer, linking the Oracle executables, and setting parameters to support Oracle operation. The goal of this process is to create a database that you can use. That seems simple enough and from my experience, this part of the installation procedure usually works quite well. (I have had to create a database manually only once and perhaps could have fixed the installer if there was more time.) By now, you will have chosen an name for your database and decided on the disk layout and all the technical factors such as block size. I say “create a database,” but what does that mean? The following are the basic steps in the creation of a database (from an internal point of view):

1. Create a set of initialization files specific to this instance (*init_{sid}.ora* and *config_{sid}.ora*).
2. Create a set of control files to store the basic database configuration.
3. Set up at least one data file and tablespace (system).
4. Set up several online redo log files.

There are several ways to create a database. The first is to select the **COMPLETE SOFTWARE/DATABASE FRESH INSTALL** menu option from the Oracle installer actions screen (see the first option in Figure 18.2). I do not like to do this because the system administrator typically is running the installer at this point, and most of the questions and configuration options that Oracle asks for on the database installation are, in my mind, better answered by the DBA. The method I prefer is to have the system administrator use the **Install/Upgrade/Patch Software Only** option. Then I, as the DBA, select **Create New Database Objects** to create the database. The final option is to create the database manually as is described later in this chapter. I do not recommend that you manually create a database unless you are really comfortable with Oracle and your operating system.

If you are using the installer to create the database, you are asked a series of questions about the data files and instance that you will be creating. If you wish to adjust the parameters in the initialization files manually, you can use your standard system text editor to create these files from the templates provided by Oracle. (In the *dbs* directory under the Oracle home directory, you will find the *init.ora* and *config.ora* default files.) Otherwise, the installer asks you about the parameters and

locations for the rest of your installation using fill-in-the-blank panels. Note that this process creates the basic tablespaces used in your instance (system, rbs, temp, tools, and users in Oracle 7). If you wish to create additional tablespaces for specific applications, you need to do this after the installation process, using the `create tablespace` command.

A few final notes about the installation process. First, review the log files (even though they are long) to see if there were any problems with your installation. Sometimes an error is recorded in the log that was not displayed on the screen or that you missed while things were scrolling by. The names on the menus may vary between different releases, but the concepts remain more-or-less the same. The only tablespace that is pretty much fixed and that you cannot move, drop, or re-create is the system tablespace. Apply special attention to ensure that you get this tablespace exactly where you want it. Finally, you may make some mistakes in the database creation process that are very difficult to back out of (all of your data files are in the wrong location, for example). Until you start loading data, it is relatively easy to wipe out the instance that you have just created and start over. Merely delete the data and log files that you created, remove the `initsid.ora` file, and remove the line from the `oratab` file.

This section provides an overview of the database creation process. Again, the exact details of this process will vary with the operating system and the release of the Oracle software. Your Installation and Configuration Guide provides the specifics. However, you should understand what is involved with the database creation process. If you plan the layout of your database (including data filenames and sizes), the actual database creation process is relatively straightforward.

DEALING WITH INSTALLATION PROBLEMS

Now that you know how straightforward the database creation process is, let's jump right into dealing with problems during the installation process. You have read all the documentation, developed your checklist, followed all of the instructions to the letter, and still receive a really ugly error message before you even start to load the software. What do you do now? The process of dealing with problems can be divided into five phases:

- ◆ Recording all the details surrounding the problem
- ◆ Determining the *real* problem
- ◆ Looking for the easy answers
- ◆ Getting help for the hard answers
- ◆ Fixing the problem

A lot of people overlook the first stage and then wind up having to re-create the problem (which sometimes takes a while) in order to figure out what really happened. This wastes time, and it's often difficult to re-create a problem. Before you leave the screen that has the error message, document the exact error message and text that you receive and on which screen the error occurred. Next, once you have returned to the operating system prompt, record all the environmental variables (VMS logicals, and so forth) that you had set up when you had the problem. Print a copy of the log file so that you have it ready. You may think that you just have a simple problem and that you do not have to spend much time in documentation—but you may be wrong.

The next phase in the process is determining the *real* problem. The error message that you receive may not be the actual problem—it may merely be a symptom. Because the installation process is not performed that often and there are such a wide range of system configurations on which Oracle is installed, the system may be so confused that it cannot determine the true error. Instead, you get something related to the true error. This is where your log files (which often contain details not found on the screen) and data on environmental variables can come in handy. Here are some classic examples of hidden problems:

- ♦ It says that something is missing when actually the path and environmental variables are not set correctly, so Oracle is looking in the wrong location.
- ♦ It says that it cannot create a file. You think that you are missing some privileges when actually you have just used an illegal filename.

Once you have determined the real problem, always look for the easy solution. For example, if you suspect that you are having problems with file permissions, set the directory structure so that any user can write to it. You can set it back later if it turns out that this was not the problem or you create the data files successfully. This is especially important with Oracle, because you may have to wait a while until a telephone support person is available. Oracle has multiple groups, installations being one of them. Within these groups, its people specialize on different platforms and you may have to wait until your expert is available. You can lose a lot of time waiting for a staff person to ask you what you had those directory permissions set to only for you to realize that only root could write to that directory.

However, there may be times when you cannot solve the problem yourself. You have diligently recorded all the information related to the problem and looked for all the obvious solutions. Perhaps you tried a few simple things to fix the problem. Now it is time to call for support. Most people call Oracle technical support; however, if you have some friends who may have just done the same installation, they could be a

quicker resource. If you call Oracle, be sure to have your customer (CSI) number ready along with the exact versions of your Oracle software, operating system, and so forth. If you do not have any idea what your CSI number is, try looking on your shipping documents. The Oracle technical support folks have a computer database (surprise!) of problems and solutions that they can search through. They need a detailed and logical account of exactly what happened (look at your checklist) and what the symptoms were (error message, log results, and so forth) to get to the solution quickly. The better prepared you are to describe the problem, the faster they can come to a resolution.

The final step is actually fixing the problem. If you are working with Oracle technical support, you may be asked to do some things you have never tried before (like running a series of make files from the UNIX prompt). Try to keep a record of what you are doing so that you can repeat the process in the future (or share you experiences with friends who will then owe you a favor). Use a new name for the Oracle log file each time you run the Oracle installer. This gives you a series of files from which you can piece together what happened.

This section explores how you should deal with problems that come up during the installation process. Obviously, it is not a detailed checklist of everything that can go wrong. There are too many Oracle versions and operating systems to make this practical. Instead, it focuses on general troubleshooting principles that can be applied across the board. Besides, Oracle support people get paid to answer your questions when problems arise, so don't feel guilty about calling them when your need is real.

MANUALLY CREATING A DATABASE

I once received a copy of the Oracle software wherein the installer could not create a database for me. The software was loaded and seemed to have linked correctly, but I kept getting error messages when creating the database. Oracle was looking into the problem and we had already spent several weeks getting patches and new distribution media. It was also a busy time because this was one of the first production copies of Oracle 7.1 that was released. Anyway, we had to get moving so I suggested that I could create the database the old-fashioned way with the `create database` command. It worked, so we never bothered to figure out how to make the installer work correctly on that release.

Creating a database is not especially difficult if you have planned everything. Basically, you need to create some initialization files and then issue a series of SQL commands in SQL*DBA to create the database itself. You need to create an `initsid.ora` and a `configsid.ora` file that specifies your desired parameters, including

the location for control files, which are created in the next command. The first command creates the database itself (the control files, the log files and the SYSTEM tablespace). Here is the format of this command:

```
create database new-db-name
controlfile reuse
logfile group 1 ('/disk24/oracle/new-db-name/log1.log') size 20K
datafile '/disk12/oracle/new-db-name/system1.dbf' size 50M;
```

When this command completes, you will have a database, but it will not be very useful. You lack tablespaces to store the user data, temporary segments, rollback segments, and so forth that are needed to have a fully productive instance. In Oracle 6, there were many instances set up with one very large tablespace. However, Oracle 7's installer creates five tablespaces by default that you will probably want (system, temp, rbs, tools, and users). To create these remaining tablespaces, use a create tablespace command similar to the following:

```
create tablespace temp
datafile '/disk01/oracle/new-db-name/temp1.dbf' size 100M;
```

You're almost home now. You are missing only the public rollback segments that users need for transactions. When you created the database, you created a private rollback segment owned by sys that the users cannot use. You now need to log in as sys (or connect internal in SQL*DBA) and create additional rollback segments. The create rollback segment command syntax can be found in Appendix A. Create at least three rollback segments for most databases and always use the optimal storage parameter. This enables Oracle to recover space from a rollback segment that has grown to a large size for a particular transaction that has completed. If you do not specify this parameter, that space will remain allocated to the unused rollback segment and not be available for future transactions except when they are using that rollback segment.

You may wish to add log files or log file groups and tweak the default storage parameters of the tablespaces that you have just created. However, at this point you have the core of a working Oracle instance. All you need is some user data and you are ready to proceed with operations. Some tasks that still lay ahead include the following:

- ♦ Adding users to the system.
- ♦ Setting up your groups and security scheme.
- ♦ Creating your application scheme.
- ♦ Running some of the SQL scripts in the \$ORACLE_HOME/rdbms/admin directory. Normally, the installer runs most of these scripts that add records and create views for Oracle internal functions, but you may have to

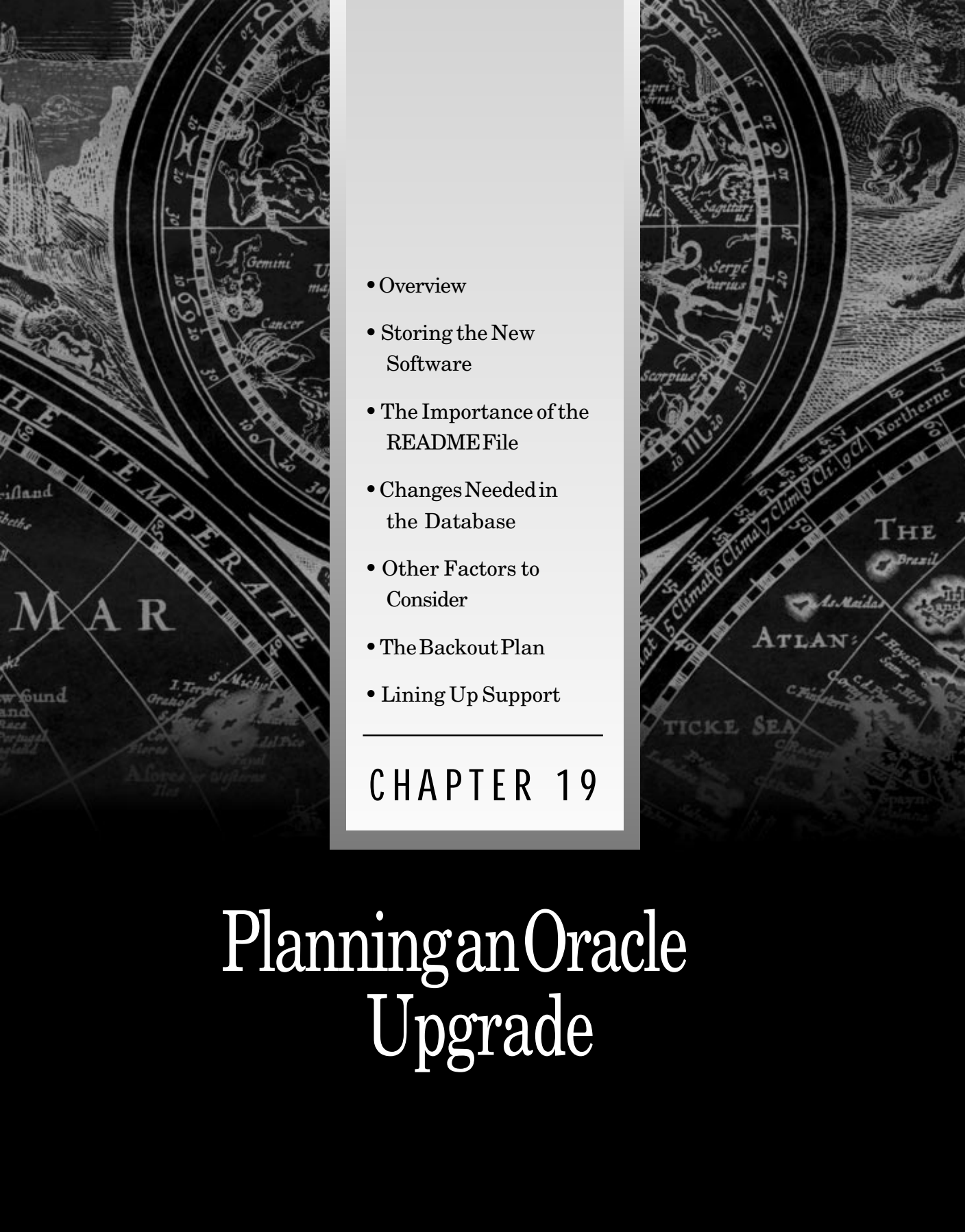
run some of them manually when you create a new database. A typical example is `catproc.sql`, which sets up the procedural option.

- ◆ Installing applications that use the Oracle database (for example, accounting systems).

SUMMARY

This chapter presented the basics of the Oracle installation process—not the details of any one particular installation, but rather the general concepts that have gotten me through installations ranging from Oracle 6 to 7.1 with a dose of Personal Oracle7 thrown in for good measure. The following list summarizes the main points of this chapter:

- ◆ The installation process is complex and the configuration of your host computer system needs to be examined carefully to ensure that it is ready for Oracle.
- ◆ Your best chance of success comes from having read the installation materials thoroughly and following their procedures—perhaps on a checklist that you have made up—to the letter.
- ◆ Plan time to deal with problems. Your installations might go like clockwork, but if not, you need time to troubleshoot, call Oracle, get new tapes or patches, and so forth.
- ◆ Good discipline in taking notes of what you are doing (checklists, and so forth) enables you to describe the problems accurately to Oracle technical support, which usually results in much faster solutions.
- ◆ Line up support from people who are familiar with the operating system (the system administrator) and, if possible, someone familiar with the development utilities on your operating system (for example, `mk` on UNIX).
- ◆ Finally, try and pick a time when you have little chance of interruption and there are no great time pressures to complete the installation. You run the risk of missing a step or making mistakes if you have to take telephone calls or you have a only half an hour to complete the installation.

- 
- Overview
 - Storing the New Software
 - The Importance of the README File
 - Changes Needed in the Database
 - Other Factors to Consider
 - The Backout Plan
 - Lining Up Support

CHAPTER 19

Planning an Oracle Upgrade

In general terms, upgrades are very similar to installations. You have a new tape or CD-ROM from the boys and girls in Redwood Shores that contains the latest release of the Oracle RDBMS or one of its tools. Perhaps it contains those fixes that your developers have been desperately crying out for or that you need to keep your database functioning reliably. Perhaps you don't know why you are upgrading other than some nebulous concept of keeping up with the latest technology. Whatever the case, you have the assignment of loading the new software on the system.

So much for talking in general terms. There is a key difference between the upgrade process and that of the initial installation—you have something to lose on your system. Yes, it is possible that you could make a mistake in your upgrade or that some bug in the upgrade process could damage your instance and perhaps that precious data that you have been so carefully nurturing for so long. Don't worry. Two really basic techniques enable you to minimize the chance that something *will* go wrong and to recover your database if something *does* go wrong. This chapter is devoted to solving the former problem—minimizing the chances that something will go wrong by carefully planning the upgrade process.

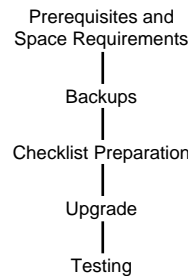
This chapter walks you through the upgrade planning process in the same fashion that you went through the initial installation planning process. You learn the key differences that you have to factor in when upgrading as well as the themes that are common to both processes. Finally, I will get on my soapbox and emphasize something else that has helped me out in the past—the backout plan. So now, without any further ado, on with the upgrade planning process.

OVERVIEW

The starting point for upgrade planning is the same as that for installations—the Installation and Configuration Guide combined with the README file for the products that you are about to install. There is no substitute for reading this basic documentation. Each release of the database and application software contains a set of changes that the upgrade process needs to accommodate. In the case of application software development tools such as Reports or Forms, you may only have to recompile your applications. This presents no risk to the data in the database. However, if there are substantial changes in the internal structure of the database (as is the case when upgrading from early versions of Oracle, such as 5 and 6, to the latest version, 7.1), you may have to export your entire database and re-create it under the new software. The exact path that you choose depends on the versions from which and to which you are upgrading. You need to read the Installation and Configuration Guide and README files to determine what is needed.

There are several elements that you should consider that are common to all upgrades (see Figure 19.1). Some of these are listed in the Installation and Configuration Guide. Others are just tips that I have used that can make life a little easier.

Figure 19.1.
Common elements for
Oracle upgrades.



The first step in this process is to go through and determine the system prerequisites and space requirements for the Oracle software again. The amount of space required has grown substantially since I first started working with Oracle and I expect this to continue as users demand more functionality and features from these products. The prerequisites portion is also important, because as operating systems grow in features, the Oracle developers try to take advantage of them. Operating systems also occasionally undergo substantial changes, as was the case of Solaris 2.3 on Sun computers. You need a version of Oracle that is compatible with your operating system release level and its supporting products (the graphical user interface, communications protocols, and so forth). Don't skip this step; it does not take that long and can save you some real headaches.

Once again, before you perform any surgery on your mission-critical production database (or even development databases for that matter), take a little time to perform a complete backup. I have twice turned a working instance and database into unreadable, unusable messes when performing upgrades. Once I made a mistake, and the other time I did what the documentation said and *really* messed up things. Even if the upgrade itself works correctly, you may find a bug with the new release of the software that causes you problems, and you may have to restore the previous release until the bugs are fixed.

Checklists during the upgrade process may be even more important than during the initial installation because you have to jump around in the book more during upgrades (in order to perform different steps for different release level changes). Again, you can either copy the pages from the book and mark up these sheets or you can type a process from scratch. Either way, it can help later on when you are troubleshooting or you have been distracted in the middle of your upgrade by a telephone call.

Finally, after reading, checking, documenting, and backing up the system, you are ready to tackle the actual upgrade task itself. I've done this two different ways, depending on the version of Oracle to which I am upgrading. The first method involved using the Oracle installer. I have seen this work only once (the good news is that it worked for the latest version that I installed). The rest of the time, I have

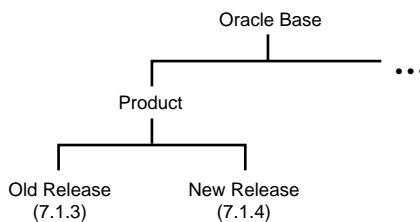
followed the README notes that indicated that I should manually run a series of scripts under the rdbms admin directory. This was relatively simple and took ten or fifteen minutes.

After the upgrade itself has been completed, run the database through a series of simple tests to make sure that the upgrade did more good than harm. One of my basic checks is to run a few SQL queries from the SQL*Plus prompt to list the tables and other objects that are in the database. Next, query a few of the key user data tables to see that the data is still there and available. Finally, let a few of the key users and developers come in to make sure that their applications still function correctly.

STORING THE NEW SOFTWARE

If you have installed the database and tools software according to the OFA, you probably have already allocated space in a directory that parallels your current software directory (see Figure 19.2). If you do not have this configuration, perhaps now would be the time to rearrange space so that you can get there. If you are installing Oracle application suites such as Oracle Financials, there will be a similar directory hierarchy for these applications. In any event, use the figures calculated from the new software's Installation and Configuration Guide to ensure that you will have sufficient disk space for the new software. You should also allow a fair amount of extra space (a few tens of megabytes) in case you need to load and apply patches to your new software.

Figure 19.2.
Location for new Oracle software.



THE IMPORTANCE OF THE README FILE

Until this last release of the Oracle database that I installed (7.1.4), I would have told you to always go by the README file for performing upgrades. In most of the Oracle 7 systems that I have upgraded, I found that the Installation and Configuration Guide said to use the Oracle installer to perform upgrades and it did not work. However, just to keep me humble, the 7.1.4 Installation and Configuration Guide said to use the installer and the README said to run the upgrade scripts manually. In this case, the installer worked correctly and the scripts, when run individually, did not.

I still recommend always reading both the book and the README file to determine your upgrade procedure. The README was developed after the books had been sent to be printed and therefore should be more current. The README also has some other interesting material, such as the final list of new features and bugs that have been fixed with this release. If you ask me which one I would follow, I would bet on the README (OK, so it didn't work for 7.1.4—it did work for all of the other upgrades I've done).

CHANGES NEEDED IN THE DATABASE

For simple upgrades wherein Oracle changes the performance features of the software but not the database structure itself, most of the upgrades seem to be updates to Oracle internal tables, views, and so forth. Between 6 and 7 a fairly high percentage of the views changed names and content to reflect the new features of the database. However, sometimes Oracle needs to change the internals as to how it stores data within the data files. The solution that it has typically used to support these upgrades is to perform a complex export of the old database and then you import it into the new database. Such changes do not occur very frequently (the last one occurred after Oracle 7.0.10); however, they can be quite painful if you have a very large (100G) database.

Another important factor to consider is which new features and parameters are supported and which have been deleted with the new release. You may have to modify your initialization files to delete certain parameters and you may want to set new ones (for example, the number of parallel server processes running under Oracle 7.1) while you are upgrading. You and your developers may have to make modification to scripts and software, so it is always best to have a test database to verify the performance of your mission-critical applications before you upgrade the production instance. At least take a look at the new and deleted features to see whether any of them are applicable to you.

OTHER FACTORS TO CONSIDER

On most production systems, you do not have the luxury of switching back and forth between releases as you explore your new software. Always have a test instance, even if it is small, to test your new RDBMS release against your production applications before you even think about trying it on production.

The new features that are available with the new release are some of the first things that you should be considering. Many of these are internal to the database and of concern only to the DBA. Bug fixes are generally appreciated by all, but you may want to remove some work-arounds that you had made to keep the old release of the

software going after you have tested the new release. You also may wish to publish the list of new features and bug fixes to your developers. They often do not get their own copies of the Oracle documentation and therefore do not take advantage of the new features that may save them time and also save database resources. I always say that an informed developer is one who is not coming to you with questions all of the time.

You should also be especially sensitive about any features that were available in the previous release, but that are no longer available in the new release. Typically, Oracle works very hard to ensure backward compatibility, but in the release documentation you will find a few features mentioned that are being phased out. You should also publish these to developers and consider removing them from your own software (admin scripts and so forth) before it becomes a crisis.

The next factor to consider is that the new version may increase the amount of memory that you use. Some of this comes about when you take advantage of new features, such as additional background processes. Others creep in when you find that you need to allocate additional space for Oracle tunable parameters to support the new software. For example, I was very surprised at the increased number of cursors that Oracle 7.1 used. We had to increase the `init.ora` file setting for cursors by a factor of 100 to get our existing applications to work under Oracle 7.1. This was not bad in that it did not appreciably increase the amount of shared memory space used, but the principle should be considered. Always check the disk and memory requirements of your new software, along with other tuning suggestions in the installation documentation.

It's always a good idea to clean up the instance before upgrading it. Perhaps it is just the mental attitude of wanting to have the instance "fresh" after the new installation, or perhaps it is the only time that you get to think about such things. Anyway, if you get the chance, clean out old, unwanted tables, indexes, and so forth. Although there is usually not a lot to do in a production instance, most development instances need this routine cleaning. Typically, I send a list of objects to the development manager and ask them to get their people to indicate which objects are still needed and which can be deleted. Some developers try to keep as much as possible, but when disk space is limited, you can use peer pressure to keep their tablespace free of clutter to support new projects.

Finally, even though the software goes through a reasonable test suite at the factory and at Beta test sites, you should always plan to run a thorough test of your applications running against the new database software before making it available to users. You may discover that you are using features that are no longer supported and therefore you have to make some changes to your application software. You may find that your application finds a new bug in the Oracle software and you have to

develop a work-around or get a patch from Oracle. Finally, your application may exceed the capacity of certain resources (like my cursors problem with Oracle 7.1). In this case, you have to adjust your tuning parameters until you get everything working happily again.

Send out the information to your developers so that they are better prepared for their development efforts. Many of these issues may not be applicable in your case. You may find that you are running only a packaged application such as Oracle Financials, and therefore you do not need to concern yourself with development issues.

THE BACKOUT PLAN

Perhaps this is another legacy of my submarine days, but I always like to have a plan in place to handle the situation wherein problems come up and I have to abandon my upgrade efforts. Perhaps it is just that I do not want to have to “wing it” when the problems come up and everyone is yelling for a solution, or perhaps I just want to assure my customer or boss that we can handle it if the software is bad or some other problem arises. This is especially important when performing upgrades to production systems.

What are some of the items that should be in this backout plan? Well, the first thing is the criteria that would cause you to have to perform the backout. It's a good idea to get management people to agree to this in advance so that they do not act like armchair quarterbacks the next day and tell you that you should have done it sooner or waited a little longer. State the criteria in terms of time (“if there are still problems at 5 p.m. and there is no immediate solution, begin backout so the system can be operational for nightly batch processing”) and other conditions (“there is a bug that Oracle needs to investigate and it indicates it may be several days”). Again, much like the installation plan, send it to managers (project managers, DBA group managers, and so forth) so that they can look at it and get their shots in before problems occur. Make them understand that if they do not give you comments, you have their approval to proceed according to the plan.

The main part of the backout plan are the steps needed to recover the instance fully so that it can run under the old software. This usually means restoring a complete, cold backup of the data and resetting parameter files, such as `oratab`, to point to the old version of the software. List all the files that need to be restored. This serves two purposes. First, you do not have to rely on your memory when it is late, you have been struggling for hours with a particularly annoying bug, and you have to restore the old instance to get production going again. Second, it serves as a good checklist for your backup that is performed as part of the upgrade plan. You really want to ensure that you have backed up all the files that your backout plan will require. This may

seem simple on small, single-disk computer systems, but when you have files scattered across 135G of disk or more, it can be a challenge. Remember, Oracle is not forgiving if you remember to restore only 95 percent of the files that it needs.

LINING UP SUPPORT

As has been mentioned frequently though the last several chapters, there can be problems with the installation and upgrade process. They can be worked through, but it takes time and it takes support from other people to make it work. Specifically, there are a few resources that you should consider lining up during the planning stage so that you can have them ready when you actually perform the upgrade:

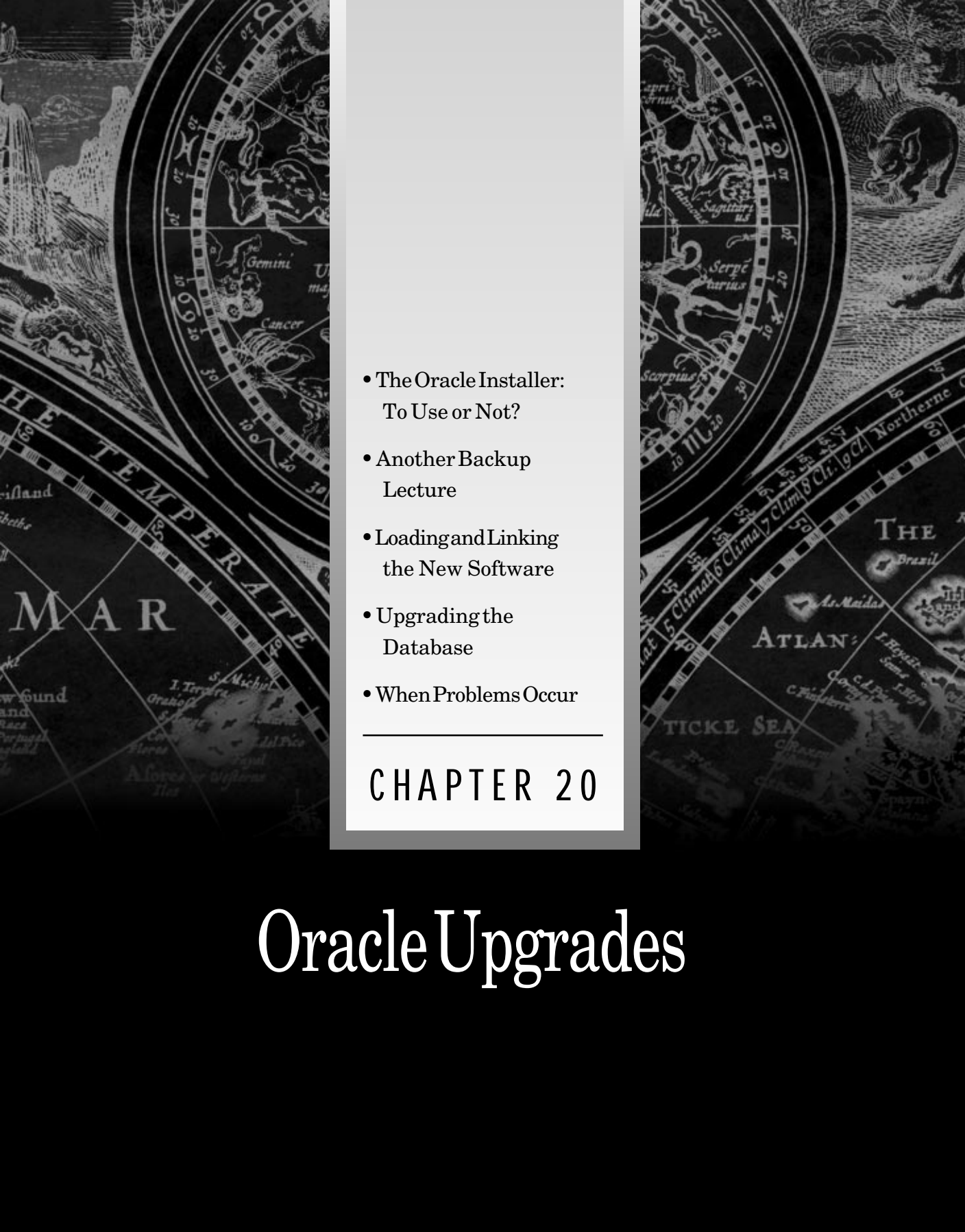
- ◆ The system administrator should be available to provide support ranging from tape backup restores to simple things such as resetting directory write permissions. If you also act as the system administrator, you can do this for yourself, but be careful to not act too quickly and to think things through. The system administrator accounts have a lot of power and therefore can do a lot of damage (recursive deletes of files when you are in the wrong directory, for example).
- ◆ If you and your system administrator are not very familiar with the host operating system, you may consider performing the upgrade during a time when you have access to operating system vendor technical support. This can come in handy when the installer is screaming that it cannot find `mk` and you do not even know what `mk` is.
- ◆ You should understand what level of service you have with Oracle technical support and perform your upgrade during the period in which you are allowed to call Oracle. Like most vendors, Oracle provides 24-hour support, but it charges you much more than the 8-hour, weekday support contract. A fair percentage of installation problems can be solved quickly by Oracle over the telephone, so schedule yourself accordingly.
- ◆ It is helpful to have a developer available to test out applications after you have completed your upgrades. In most of the instances that I have worked on, I knew the data structures and basic processing flow, but I was not a user and therefore was not familiar with navigation between panels.
- ◆ Finally, it is helpful to have a shell script programmer available in case you have to perform functions such as manually linking the software. Usually you can do this with support from Oracle over the telephone, but you may need to find someone locally who knows all the tricks you need to do, such as setting up your path and environmental variable to access these development utilities. System administrators are familiar with the general concepts, but they usually do not work with these utilities regularly.

SUMMARY

So much for the upgrade planning process. In many ways, it resembles the installation planning process discussed in Chapter 17. Some of the key differences include the backout plan and the need to align resources to support a quick upgrade. You usually cannot leave a system half upgraded for several days while you work out a list of bugs. As always, have a test instance and upgrade that instance first before you go anywhere near the production instances. You may well find new glitches that are unique to your local applications, and you want to have time to work out these problems. Once you perfect the upgrade procedure in test, you can usually use it again in production. Because production upgrades seem to attract the most management and user attention, it is always better to make it look like a smooth process that you control.

Document your upgrade procedure and a backout procedure. The upgrade process in the book is somewhat difficult to follow at times (you do three steps on one page and then skip two pages back, and so forth). It is easy to lose your place when the telephone rings or someone comes into your office. The printed checklists enable you to mark what you have done and any problems that you have noted. This marked-up checklist is very useful when discussing problems with Oracle technical support. Finally, a written backout plan is useful to ensure that you correctly restore the old software and database in the event that you cannot complete the upgrade.

Obviously, the level of detail for your planning will be determined by the importance of the instance that you are upgrading. You may feel a little freer working with a test instance that you have created for yourself as opposed to the most important production database in your company. I, however, like to use the test instances to prove that my procedures will work before I go to production.

- 
- The Oracle Installer:
To Use or Not?
 - Another Backup
Lecture
 - Loading and Linking
the New Software
 - Upgrading the
Database
 - When Problems Occur

CHAPTER 20

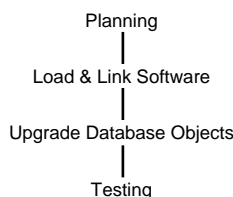
Oracle Upgrades

This chapter presents an overview of the Oracle upgrade process. As with most of the chapters in this section, you need to look at the documentation (the Installation and Configuration Guide combined with the README files) to determine the exact procedure that Oracle requires to complete the upgrade. This chapter captures an overview of the general flow of procedures that are typical of Oracle upgrades and then provides a few tips.

The actual upgrade process may take days if you need to perform an export/import from an old version of the Oracle software to one of the new versions. However, if you are performing a relatively minor upgrade between recent versions of the software, it can take less than 15 minutes to upgrade your database once you have loaded the software on your system. The basis for the entire process is the planning discussed in the last chapter. The plan should take only an hour or two to develop and can save you many hours of problems.

When you complete your plan, load the software onto your disk drives. After the software is installed and linked, there are typically three paths that can be used to upgrade a database to work with the new release of the Oracle software (see Figure 20.1). The first is to use the Oracle installer to perform the upgrade. I have been able to use this only twice in the upgrades that I have performed. Typically, the release notes override the Installation and Configuration Guide and tell you to follow a second path and run the upgrade scripts in the rdbms admin directory manually from the SQL*DBA utility. The third path that you may have to follow is to perform a complete export of the old database, possibly followed by some special migration utilities, and then restore the database to your new instance. These options are discussed in a little more detail later.

Figure 20.1.
Typical upgrade
processes.



Always consider some basic testing of the new instance as an integral part of the upgrade process. This way, you can find any problems before someone else does. Others may be too busy to give it a thorough test. If you do the testing, you find out about the problems before they reach crisis proportions. A few simple queries are a good way to check basic instance functionality.

THE ORACLE INSTALLER: TO USE OR NOT?

Read the Installation and Configuration Guide along with the README file to determine whether you should use the Oracle installer. The Oracle developers make the rules in this case—you and I do not get a vote. So don't try to wing it or try something imaginative. There are too many internal details related to the upgrade over which you have no control.

Part of almost every installation uses the Oracle installer. You still have to transfer the software from the distribution media and link the executable files together. You use the same menu you used before, but this time you select the Install/Upgrade/Patch Software Only option. You follow the prompt screens and, with some luck, you have a working set of application software.

Recall that Oracle data upgrades fall into three categories. The first category uses the Oracle installer. This is relatively simple; set your `_` environmental variable to point to the new software and run the `oranist` software that was supplied with your new release. This time you select the Upgrade Existing Database Objects selection on the Install Actions menu. This takes you through a series of prompts. After this is completed, you should have a working instance running under the new software. This is the clean, simple, menu-driven way to upgrade the database.

However, history shows that the installer is not always ready to perform the upgrade correctly. In these cases, the README file directs you to run a series of SQL scripts that are located in the `rdbms admin` directory. This is usually relatively straightforward. You access `SQL*DBA` and connect internally (which gives you access as the `sys` Oracle user ID that owns all the internal database objects). You then execute the scripts using the `@` format. These scripts actually call a series of other scripts, but the whole process still usually takes only 10–15 minutes. When completed, you should have a working Oracle instance running under the new release of the software.

The final path applies only when you are performing an upgrade in which Oracle has changed something about the internal storage format between your current release and your new release. The current software on the market (Oracle 7.1.4) requires significant upgrades only when you are upgrading from the relatively old releases of Oracle, such as 7.0.10 and earlier. The basis of this process is that you export all your data to the neutral Oracle export file format and then import your data into an instance running under the new software. The exact process followed in these cases

is specified in the documentation. You may even have to run certain special upgrade scripts in addition to using the export/import utilities. This process could be challenging and, if your instances are large, time-consuming. As always, secure a reasonable amount of time to perform these upgrades and leave room to deal with problems that may arise.

ANOTHER BACKUP LECTURE

This is yet another backup lecture, but I will keep it brief. I discussed backup in the upgrade planning chapter, but some people probably skipped directly to this chapter because they are never given enough time to plan. You have two goals when performing your pre-upgrade backup. First, the data has to be consistent. Use cold backups whenever this is practical. Second, your data has to be complete. You need all the data files, control files, log files, and initialization files associated with the instance to be ready on tape in case they are needed.

LOADING AND LINKING THE NEW SOFTWARE

As stated earlier, this is the exact process that you go through when performing a new installation of the software. However, for those of you who have not gone through an initial installation, but have to upgrade a system installed by someone else, consider the following. First, you need to read your installation documentation to determine the set of commands that you need to get access to the installer. For tapes, this usually means copying the installer and its related files using standard tape commands. Then you run the installer to load the rest of the software. For CD-ROM installations, you can usually run the installer directly from the CD.

You will be asked to choose which components of the Oracle software suite you will be loading on your system. Oracle typically designs its distribution media to contain a standard bundle of software, some of which you may not have licensed or may not want to install on your system (it can take a lot of disk space). This is where the planning process that you went through comes into play. You select components based on what you will be using combined with the prerequisite analysis that you completed. Many products (such as the Oracle toolkit or GUI common tools) are not used directly by the users, but the tools that the users need depend on these utilities.

From here, Oracle asks you a few questions about where to put the software and some key system parameters, such as what is the DBA group called (it is usually defaulted to dba). After this, Oracle analyzes the products that you have selected to ensure that you have met all of the prerequisites. If everything checks out, it begins transferring the application software to your system. When it completes the transfer, it links the executables of the rdbms and tools.

Finally, you get a happy dialog box that tells you that the process was completed. Note that it reminds you to check the installation log to see whether there were any issues that you may want to consider. Typically though, all errors are displayed on the screen for you and the log provides only amplifying details. Save the log file in one of the DBA admin directories for future reference.

UPGRADING THE DATABASE

Once the software installation is completed, it is time to upgrade the database. Again, based on your reading of the documentation and the versions that you are upgrading between, a path is set for you to complete the upgrade. Follow the checklist and make notes along the way when you see different messages on the screen. If there are problems, you may want to be able to read from these notes (especially if it is late at night and you have been fighting your system for 12 hours). Again, this process can be a little tricky as you jump between sections of the documentation. Either type up a procedure or copy and mark up the procedure from the Installation and Configuration Guide.

WHEN PROBLEMS OCCUR

If you perform a number of upgrades with the Oracle software, odds are that you will run into some problems. Let's hope they are relatively minor and can be fixed quickly. The following are a few observations about dealing with upgrade problems:

- ◆ Upgrades, especially those on production systems, are usually time constrained. However, you should take time to document the exact error messages and the exact environmental variable (–, –, and so forth) that you had in effect when you ran into a problem. If you have to go to Oracle technical support, solutions come much faster when you have all your data assembled and you do not have to walk through the process again to get the data that technical support needs.
- ◆ Given the time constraint on most production (and even a high percentage of development) systems, you need to keep an eye on the clock to determine whether you need to back out the upgrade. Users and management are more sympathetic to your problems when you back out the changes and do not interfere with production.
- ◆ Always take a few seconds to try to determine what the real problem is. Do not assume that the error message is the actual cause and not just a symptom. For example, if the installer under UNIX tells you it cannot find –, it could mean that the make utility is not installed or it could mean that your path is set incorrectly.

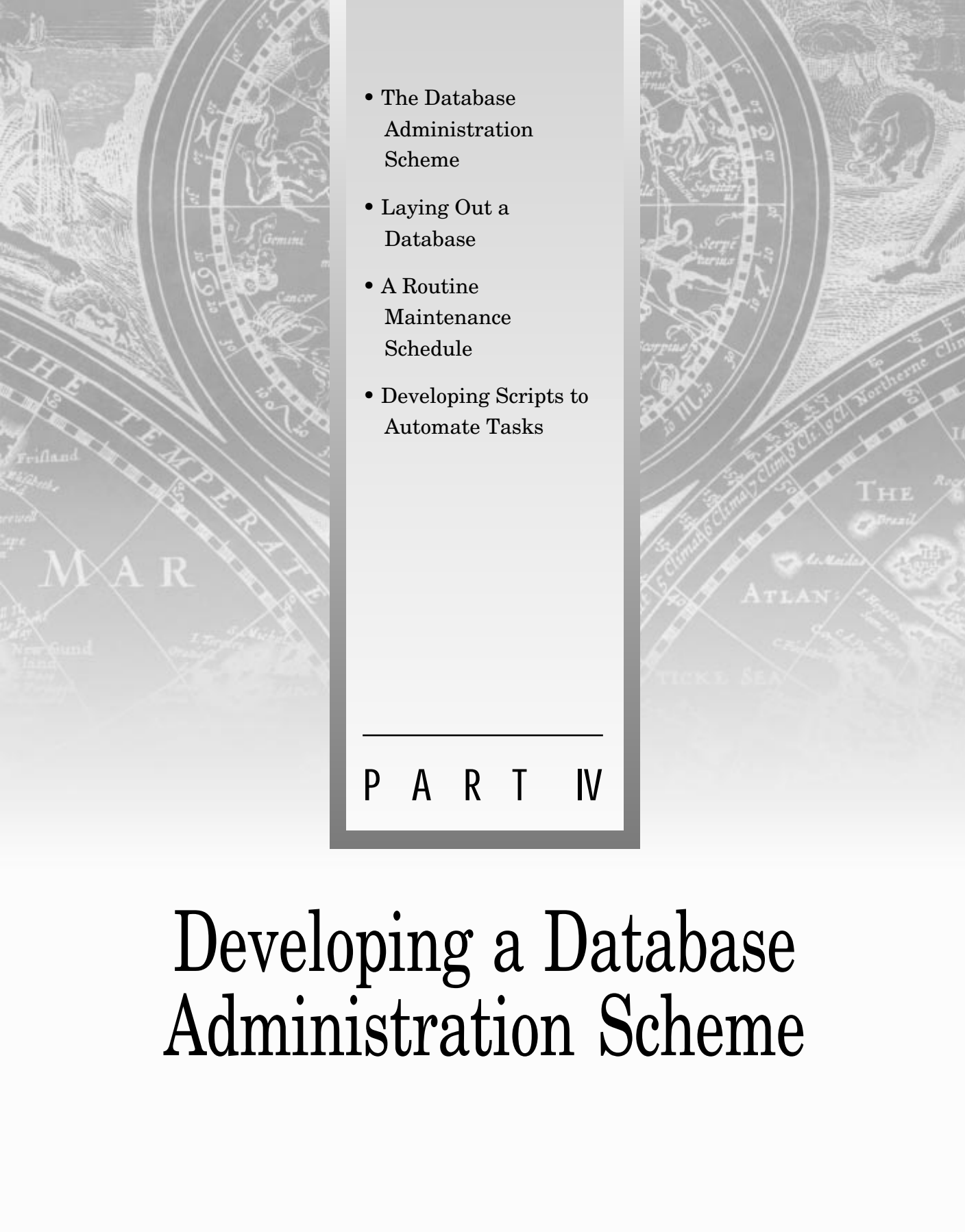
- ◆ Line up system administration and Oracle support resources when you perform the upgrade. You need to be sensitive about the hours of support that you have under your Oracle support contract.
- ◆ Finally, try to relax as much as possible. Spend a fair amount of time planning and realize that there are some things beyond your control (such as bad tapes, for example). DBAs who try to rush or perform major evolutions without proper planning time are much more likely to make mistakes that can take a fair amount of time to correct.

SUMMARY

As with the installation chapters that came before this one, you need to look at the Installation and Configuration Guide and README files that are specific to your operating system and the release of Oracle that you are trying to install. The procedures vary between releases and operating systems so do not try performing an upgrade based on the wrong documentation. This chapter provides an overview of the entire process. Too many times people start reading through the steps in the manual without figuring out where they are going first.

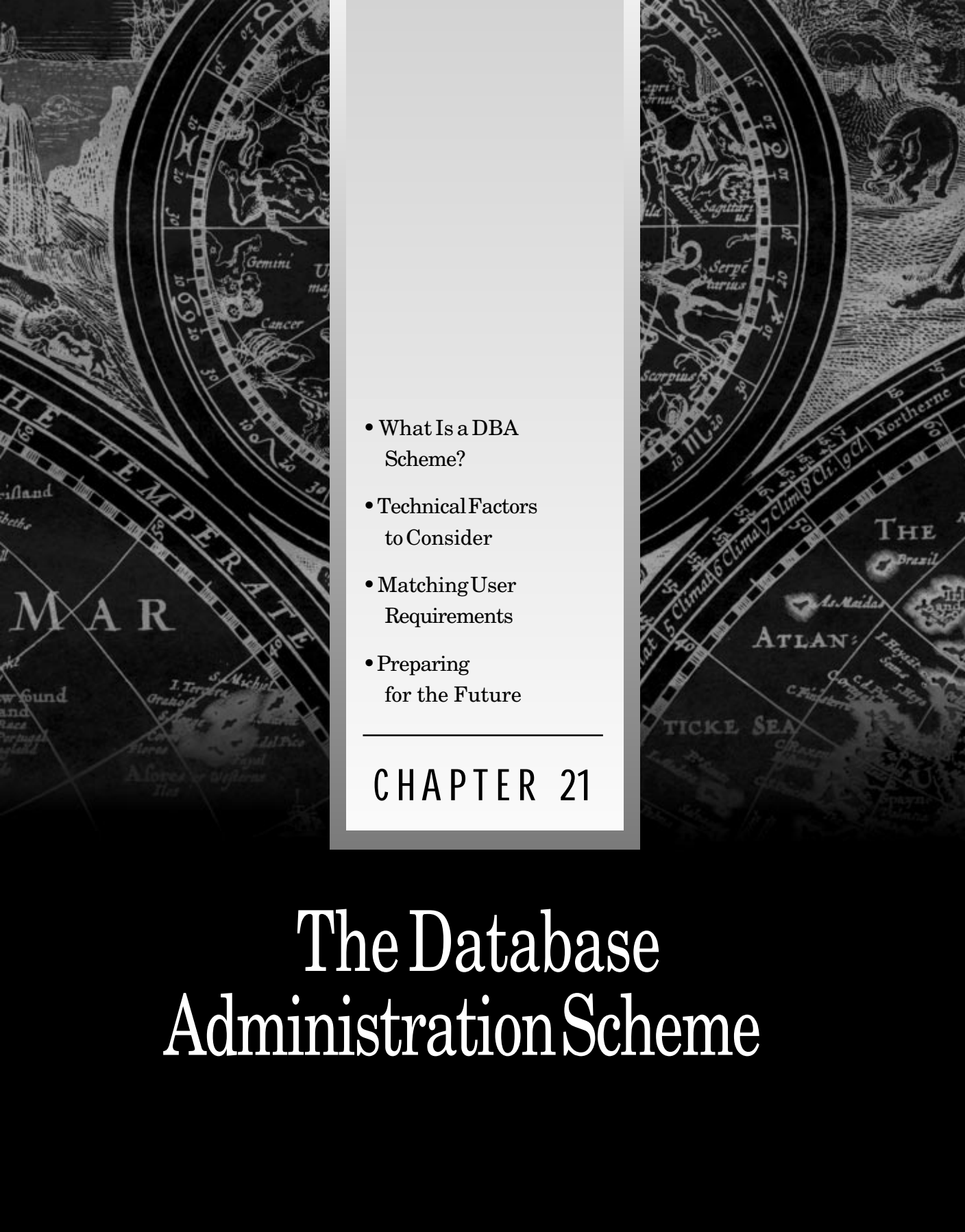
This section was somewhat difficult to write. Much of this feeling comes from the fact that I have experienced bad media, installer problems, and numerous other problems during the installations and upgrades that I have performed. It is a task that we do not perform very often, so we never quite get comfortable with it. However, with planning, checklists, and a little help from Oracle technical support, I have always managed to get the instances working.

Now that you have covered the technical foundations so that you understand your new system and have explored installation, it is time to move on to more interesting material. The next part of the book explores a database administration scheme—a plan that covers how to approach the job of being a DBA on a particular system. Perhaps you do not want to write one up and publish it. However, the issues that arise when constructing a plan should be considered by all DBAs.

- 
- The Database Administration Scheme
 - Laying Out a Database
 - A Routine Maintenance Schedule
 - Developing Scripts to Automate Tasks

P A R T IV

Developing a Database Administration Scheme

- 
- What Is a DBA Scheme?
 - Technical Factors to Consider
 - Matching User Requirements
 - Preparing for the Future

CHAPTER 21

The Database Administration Scheme

At this point in the book, most of the preparatory work is out of the way. The introduction to database administration in Part 1 was designed to give a general flavor for the job. The technical foundation work in Part 2 was designed to enable you to understand what you are working with and why some things work the way they do. Finally, the installation section gave you some insight into the process that you have to go through to get the Oracle software installed on your machine and ready for action. Now it is time to move on to the actual work of the DBA.

The description of the DBA's work begins with a name that I created on one of my recent consulting assignments—the database administration scheme. The goal was simple. This was a group new to Oracle (coming off of a mainframe) and I wanted to capture on paper and obtain agreement on how the database was going to be managed and how to set up the accounts and applications. I had been on a number of engagements where an extensive amount of rework was being done to clean up the security scheme and organize the files for efficiency after the applications had already been developed. This seemed like the opportunity to avoid rework by doing things right in the first place.

Most of the components of this scheme have been around for years. You see them in data center or software development group policies, procedures, development standards, and so forth. However, there are many computer shops out there that are very sensitive to the overtones of the word “policy.” The DBA is not always allowed to provide input to development standards (which are made up by the development group). So I chose a word (“scheme”) that seemed relatively unthreatening (probably because very few have misused it yet). It describes a series of standards related to the administration of the Oracle databases that enables the users and applications to get the most out of the Oracle database yet provide a “good enough” security infrastructure.

There are two key components to the last sentence. First, getting people to work with the Oracle database. There are many ways to write an application. Many developers tend to code their applications with application security, the way they did under older flat file systems. You could make it work that way, but Oracle has more efficient and secure mechanisms if you are willing to learn and use them. The second part is the “good enough” security infrastructure. Most organizations do not face the complex security problems of the NSA or CIA and do not want to spend the time or suffer the inconveniences that users of top secret data do. The key is to provide security that meets requirements yet also provides functionality and service to the users.

WHAT IS A DBA SCHEME?

A DBA scheme is a compilation of good ideas on how the Oracle RDBMS will be used to build and service database applications. That's a rather broad statement. There are a lot of detailed technical issues that have clear answers. For example, the

configuration of your rollback segments is determined by performance needs of the application. There is no need to write a policy about such detailed technical issues. There also are a number of organizational issues and support issues that are typically the purview of management. These issues are not included in the scheme. A DBA scheme should include answers to issues wherein there are a number of alternatives that will work, but you have to choose between them. The following are some examples of sections to include:

- ◆ Overview of how the databases are to be used
- ◆ General database configuration
- ◆ Data security set up for the instances
- ◆ User access
- ◆ Database object creation, ownership, and naming standards
- ◆ Routine database monitoring and tuning services provided
- ◆ Problem resolution
- ◆ Advanced planning

The overview does not have to be long-winded and may seem obvious to some people. If you have only a single database, this section tends to be rather broad. However, I have run across people who did not know that they should not put a certain type of data on a particular computer. Perhaps some data is sensitive, such as personnel information. Perhaps you just want to keep certain disk-space-hogging groups on their own computers so they do not have an impact on others. It is useful to lay out what applications, data, and users will exist on a given computer. Included in this is whether there will be any development or ad-hoc data stored on the computers. Some power users like to extract data from large tables and play what-if scenarios on their own tables. It is also important to describe the hours of availability of various systems, because you need time to perform some of your maintenance activities.

The general configuration of the databases is an education for both users and developers and a little discipline for the DBA. It makes you think out where you are going to put things in advance (what tablespaces, which schemas, and so forth). It is an education for users because they get a feel for the physical side of storage. Some even think to check how full disks are before they ask for new space. Just keep it to the basics of what the database stores, which disks it uses, and a general idea of the sizes needed.

Data security needs to be discussed for each of the instances. This includes how access to particular objects is assigned (roles, individual users, public, and so forth) and how the users will obtain access to the data. For some instances, this may be assigned based on the job description or group to which the users belong. In others, there may be some formal procedures to obtain access. Whatever the case, it is a good idea to have this documented so that everyone follows the same rules.

Related to data security is user access rules for the instances. Most user groups would like to establish accounts and access for their new staff quickly. However, there needs to be some control to ensure that privileges are assigned correctly. For example, there may be a central security administrator who controls access to the various computer systems. This is one of many cases where you have to adapt to your local organizational climate.

If you are supporting development, database object creation, ownership, and naming standards become important. When you have controlled applications, the ways users can add data are controlled and they use the computer only to perform necessary business functions. Developers who can create database objects have amazing capabilities for filling up disk after disk with test data, copies of tables that they do not want to lose, and so forth. Because they can easily exceed the capacity of most systems, some form of control is needed over how much space they allocate and where they put their objects (yes, they will create things in your system tablespace if given the chance). In addition to capacity quotas, you need to consider object names. This is especially true of permanent table and synonym names that developers may hard-code into their applications and with which you may become stuck. I generally like to establish a table naming hierarchy and have developers create public synonyms for all the data resources that may be moved into production. Try to get all applications coded to look for the public synonym name versus a true schema.table combination. Then there is leeway to reorganize and rename the basic table data without interrupting the applications (just ensure you set your new public synonyms correctly).

One of my favorite topics is the routine database monitoring and tuning services that the DBA group will be providing. In a sense, you are advertising some of the services that you will be providing that others may not be aware of, especially if the database continues to function without problems. Consider what level of service you can provide and document it for management. (Who knows, perhaps someone will even appreciate it some day.)

Problem resolution is an important issue that needs to be discussed, and the resolutions need to be understandable and available for everyone. Your detailed procedures and what to check are a little beyond the scope of discussion and will evolve over time as you become a real hotshot DBA. However, topics such as the priorities of various types of problems and whether they merit your working weekends and evenings on them are useful. Perhaps this is already covered by your data center policies, but if not, it may be useful to discuss them here.

Finally, I always like to include a section on advanced planning. Perhaps it is because I see so little of it. The unfortunate truth is the computer industry is undergoing such rapid changes in organizations and technology that too much seems to happen in a crisis mode. I acknowledge this, but I still like to make an attempt at planning. Typical activities for the DBA are an annual disk storage

capacity planning exercise that occurs in time to be factored into annual budgets. I like to take the results of the monitoring activities and factor in any new projects that the current numbers do not reflect to calculate total capacity requirements. You also need to consider computer processing and memory capacity needs when you perform your advanced projections. The area of capacity planning is rarely precise and subject to many errors, but it beats nothing at all.

These are some of the things that you should consider for inclusion in your database administration schemes. Each DBA needs to understand enough about the quirks of his or her organization to determine whether there are other hot buttons that may need to be considered in this planning exercise. The format can be a pretty, formal document, briefing slides, or even notes to yourself on a sheet of tablet paper. The exercise of considering how you want to do things is important. It is worth the time spent, although finding that time is sometimes a challenge.

TECHNICAL FACTORS TO CONSIDER

Life is not all interpersonal and political considerations. Actually, once of the goals of a database administration scheme is to merge the organizational concerns that you have to deal with (hours of availability, what developers are allowed to do) with your understanding of Oracle's technology to ensure that people get what they want. This is often no small task. People with far more experience and seniority than you may have been doing things a certain way since the days of punched cards. Your organization may have certain things "cast in stone" regarding the way applications are developed and data is stored. Your job is to merge the goals of these old methodologies—which usually made good sense for the tools that were used at the time—with what it takes to work efficiently in Oracle.

What are some of the old ways of doing business that are not in line with the most efficient ways to use Oracle? The first one that comes to mind is having a separate data file for every need. Oracle tends to group data into one or more tables that are organized to split data loading and provide logical groupings. Many file management systems in the mainframe world gave developers their own little world to manage. They will be uncomfortable having to give up control to someone else regarding the structure and location of what used to be their data files. Whatever creation privileges you choose to give your developers, only DBAs in Oracle will have permission to create additional data files to enable more information to be stored on the system.

Another issue is the design of the tables themselves. People tend to design tables in the new Oracle system the way they designed them in the past. If the past was a flat file system, you will find massive tables with a large number of columns, repeated data, and so forth. Although this will work, it may be impossible for you to get adequate performance out of this design, especially if you go to larger systems.

Remember, these large file structures worked for batch processing routines that provided very limited functionality on their reports. People will probably want to use the new system on an interactive basis, where response times need to be measured in seconds. Therefore, be careful and helpful when it comes to reviewing table designs or you may be in line to hear the infamous “the database is too slow” cries once the applications go into production.

A related issue, once you get developers to start developing normalized tables, is that of referential integrity. In flat file systems, data can be repeated as many times as desired. In normalized tables you have to be concerned about having good, sound keys to link the tables together. You get confusion if data in a lookup table is repeated multiple times (Oracle is not sure which value to use). Again, this is something that DBAs are often called upon to provide assistance for, because many developers come to the DBA thinking it is a bug in Oracle when they get the “query retrieves multiple rows” type of error.

While we are on a roll here with development technical issues that you need to consider, let’s look at where to put security. Most older systems relied upon the applications to control some or all of the security for the data. Perhaps there was a mainframe security package such as ACF2 or RACF that provided another level of control. You need to be very familiar with the security mechanisms provided by Oracle for data access and get some agreement as to whether these tools will be used. I recommend using Oracle security wherever possible because it provides a single maintenance point, which is desirable in any security scheme. You do not have to worry about modifications to your software applications deactivating your security scheme. This is an issue that you should bring up, even if you do not have the final say in the matter.

Another technical issue that you need to consider when planning out how your database will work is where the sorting and selection of data will occur. This applies only to the client-server world, but can greatly affect performance. You have basically two extremes here. You can perform all the sorting and selection on the Oracle server and transfer only the data you need down to the client. Conversely, you can transfer all the data down to the client and do all the processing there. I have typically found that most installations perform the sorting, selecting, and calculation on the server, because the network transfer rates limit your ability to transfer very large amounts of data. However, there may be cases where you have very fast PCs and networks combined with an overloaded server where you can adjust loads to improve performance.

There are many other issues that can come up when you are trying to plan how you will run your database. Some may be unique to your organization or driven by bad experiences that you have had in the past. Feel free to make this administration scheme planning process accommodate your local needs. Before leaving the technical section, consider the following thoughts about how to present the technical issues to your users, developers, and management:

- ◆ Much of the computer industry is extremely sensitive right now. People feel threatened by new technologies replacing stuff that they have worked with for years. The continuous threat of layoffs, out-sourcing, and so forth has increased tension levels. Your challenge is to tell people how to do things that will work best with Oracle without threatening their pride or sense of control over their worlds. It is not easy.
- ◆ You may have to deal with many years of bad attitudes towards central computer support organizations. For a long time, many data centers bullied their users and developers. Rules seemed to be based on logic such as “because I said so.” The departmental computer and desktop “revolution” (as some people have called it) has left a lot of people feeling a little too free for my taste. They want to have no standards and do not care whether they interoperate with anyone else.
- ◆ Finally, you may be dealing with a lot of people who are new to the Oracle, relational database, and client-server technologies. They may argue with you just because they do not want to admit that they don’t understand what you are talking about. Perhaps they have been told to get a pilot application ready in three weeks and they feel they cannot design the tables in this new, normalized way and still get done on time.

MATCHING USER REQUIREMENTS

The last section discussed technical issues that you will have to handle. Perhaps you are a techie who is very comfortable with issues of technology—the bits, bytes, SCSI controllers, and so forth. One of the trends that I have observed recently is that even data center techies are being required to be sensitive to user needs and business processes. In the old days, when batch programs ran at night producing reports, all you needed to know was that you had enough disk space and processing time to get the scheduled jobs done. Now, however, with online systems, user demand can vary greatly based on the business group cycles, special projects, and so on. Gruff technologists now have to become sensitive facilitators to help their business people get the job done.

One of the first issues that usually comes up is the debate over hours of availability on the system. If you are just coming off a mainframe-based system, you may be lucky for a while. People are used to having the online system shut down some time during the evening so that you can start batch processing. They will soon realize, however, that your new servers are sitting idle most of the evening and weekend. In their defense, these users are probably facing the same more-or-less pressures that the computer groups are. They may now have to work weekends and evenings to get their jobs done and they want their computers. Your challenge is to perform the diplomatic task of giving them as much availability as possible without compromising the time that you need for sound database administration tasks such as backups and upgrades.

Once you have settled the general question of the hours that the computers will be available to the users, you have to face the issues related to the business processing cycle. Some databases have the same amount of work to do every day. However, most will find some cyclic variations, such as monthly closing of the accounting books and special processing just before a major delivery. There are two things that you need to work on here. First, you need to try to accommodate these user needs. Perhaps you can offload certain tasks on days when special processing will be occurring. Second—and this is sometimes the hardest task—is to set up so that the users tell you in advance when they will be doing exceptionally heavy processing. Some things, such as the routine closing of the books, you can predict (the first four days of every month, for example). Sometimes, however, the first time that you find out about some special processing is when your system grinds down to a snail's pace and everyone calls you about performance. That's what the database administration scheme is all about—getting you to think about these issues and get processes set up in advance to make your life just a little easier.

One final note about sensitivity to user requirements: Developers are users, too. They have delivery cycles and special needs just like the users in the business units. It is sometimes easy to assume that because they are full-time computer professionals, they should know better than to start massive jobs without telling operations or the DBA. If they need the system late at night, they should tell everyone involved who may be performing other tasks such as database maintenance. Things move along more smoothly if everyone agrees to coordinate special requirements with one another and share schedules that include important milestones such as product deliveries.

PREPARING FOR THE FUTURE

Database administrators and system administrators are the ones who have the longest lead times to deal with changes. This is usually reflected by the fact that developers can rewrite large amounts of software and business people change their business processes relatively quickly. Some changes have no effect on the computer systems, but others can have large consequences. Some of the most difficult to deal with involve requirements for disk storage space, processing capacity, and memory. These items often have long budget cycles and acquisition periods. Therefore, it is worth the effort to solicit data on changes that your users and developers are planning as far in advance as possible to ensure that you will have the capacity to accommodate them. As unfair as it may sound, DBAs often get blamed when system performance degrades because, for example, a large number of new users are added and the system's memory capacity is exceeded.

The future planning process is not an easy one. There may be changes beyond your control or even the control of your users (industry trends, new regulations, and so forth). However, if you have at least a basic planning process, you are usually in a

better position to deal with these changes. It helps to have a little spare capacity tucked away to see you through until your budget and acquisition process can get you the upgrades that you need.

Formal efforts and tasking to produce capacity plans are the cornerstone of future planning. You should start with the results of your monitoring activities to track how disk, CPU, and memory usage has grown over time. It is usually more difficult for budget types to argue with hard figures that show utilization growing steadily over time. You need to be sensitive to the fact that Oracle allocates extents and data files in chunks. Therefore, you may not grow for several months and then suddenly start growing again. Your job is to explain in simple terms why this happened when you present your figures.

The next future planning technique that I like to follow is promulgating any new database-related technologies that I come across. I tend to get a large number of magazines on all subjects. However, I do get some special Oracle-related ones that my developers and users typically do not get (they probably like to read journals on C programming or accounting). This enables them to review material in a non-threatening manner (you are not proposing it in a meeting in front of their management). As long as I do not overdo it, it helps stimulate technical discussions and cooperation with many of the developers that I have run across.

Another future planning technique that you may not have considered is table and data file architecture. If you organize things logically from the start, it usually makes it much easier to move things around. When you are installing new disks, leave extra space for those data files that have a history of steady growth. Try and position yourself so that you can anticipate when you will need additional data file space and have a location waiting for that need when the time comes.

Finally, there is the sometimes sensitive issue of the margin of error. Everyone knows down deep that there will be unexpected needs that arise, but not everyone is willing to stand up and justify that before budget reviews. This can be a challenge in the database world, because changes in business volume (for example, sales are good) and other such factors can eat up disk space rapidly. Many of these trends are impossible to predict. Therefore, always leave some extra disk, memory, and processing capacity available when you perform calculations of what you need. Do not confuse this with hoarding large amounts of disk space and thereby wasting company money. The trick is to find that right amount that will enable you to look good when unexpected needs arise—somewhere between 10–30 percent is a good rule of thumb.

Planning for a future that you do not control and cannot predict exactly is difficult. There may be other factors that affect your planning efforts. Some government organizations have extremely long lead times for major equipment purchases (years). These organizations need larger margins of error than a small, cash-rich

company that can buy disks in a matter of weeks. You may have to deal with personal opinions and hot buttons from people in your management chain. Future planning can be a dirty job, but someone has to do it.

SUMMARY

This chapter presented the database administration scheme, a concept that involves the middle-level issues associated with a database. There are high-level issues, such as which vendor's product will be purchased and whether to develop or purchase applications. There are also low-level issues, such as how large are you going to size a particular table and in which tablespace an index is going to be located. These issues need to be dealt with separately. The high-level issues are usually the prerogative of management. The low-level issues are usually dealt with on a day-to-day basis. The DBA scheme forces you to think about how you will take the upper-level guidance and translate it into a scheme that can guide you in your day-to-day tasks.

Whether you implement a formal plan or reject some of the suggestions, at least consider them and settle in your own mind how you are going to approach these issues. The following is a sample outline based on some of the database administration schemes under which I have worked:

Database Administration Scheme

1. Introduction
2. Overview of the Databases
3. Security Scheme
4. Object Creation and Ownership
5. Disk and Memory Utilization
6. Routine Monitoring
7. Capacity Planning
8. Problem Resolution Procedures
9. Service Requests

Appendixes

- A. Service Request Form
- B. New User Account Request & Approval
- C. Problem Report

- 
- Overview
 - Data Files
 - Control Files
 - Online Redo Log Files
 - Archive Log Files
 - The Configuration Process
 - Expansion of the Database

CHAPTER 22

Laying Out a Database

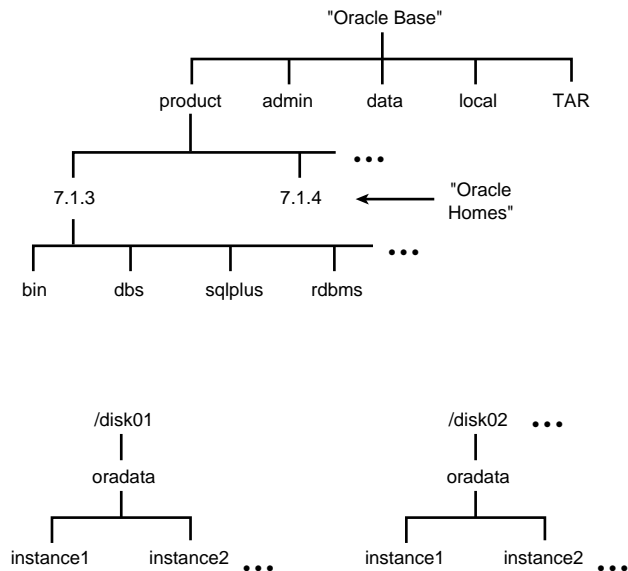
This is another one of those topics that could probably fill a book, but we have only a chapter here. There are a few sound principles that apply to the vast majority of databases. This chapter introduces you to these basic considerations so that you can lay out your database in a sound manner. I hope you read this before you have spent a lot of time configuring your system. It is always easier to build a system correctly from scratch than to rearrange things later or when the database is in production and there are only limited times for maintenance.

So what exactly does laying out the database mean? You have a large number of files associated with Oracle itself and your data. There are many possible ways to arrange these files on the disks that you have. Some ways work better than others. The goal of this chapter is to provide you with some of the considerations that you want to factor in when you plan how you will arrange your databases.

OVERVIEW

Recall from Chapter 8 the overview of Oracle's Optimal Flexible Architecture (OFA). Figure 22.1 shows the structure that I have run across under UNIX, VMS, and other common servers. The configuration under Personal Oracle7 for Microsoft Windows is somewhat different, but in this case, you are usually limited in the number of disk drives over which you can arrange your data. Therefore, in the PC case, you can usually accept the configuration defaults.

Figure 22.1.
The Optimal Flexible
Architecture (OFA).



The OFA does a good job of providing space for the Oracle products. Different releases and different products are neatly stored in their own directories. There are a number of directories for the DBA to store log files, initialization files, and so forth.

Two important ones are `ORACLE_BASE` and `ORACLE_HOME`. `ORACLE_BASE` is the high-level directory under which all the Oracle software and administrative directories are located. `ORACLE_HOME` is the location under which all the files associated with a given release of the software (for example, Oracle 7.1.4) is located. The OFA goes further than merely providing places to store Oracle software and administrative records. There is even a directory to store your data. The OFA also creates directories for each instance's data and log files. The standard documentation discusses a plan for you to incorporate multiple data directories, each with its own name and subdirectories, for each instance that uses them to store data.

So far, so good. It is a relatively simple and clear scheme. You have plenty of tidy little boxes in which to store your precious data. It is a workable scheme, and the Oracle installer helps you by using this scheme as a default once you supply the Oracle home and Oracle base directories. If you have a single disk drive on your server, your configuration planning is pretty much completed. You can accept the defaults and be done with it.

However, suppose you have multiple disk drives on your system. Suppose also that you are trying to squeeze every ounce of performance out of your disk storage system to maximize the performance of your applications. Many of the large systems that I have worked with have been limited by their data transfer capacities rather than their memory or processing capacity. So now you rise up to the challenge of not just sticking the files where they will fit. Instead, you are going to arrange them to balance data transfer loads and maximize redundancy and reliability. Here is a simple three-disk example for file distribution:

disk 1	Oracle software itself
	Oracle log files
	Rollback tablespace
	Temporary tablespace
	System tablespace
disk 2	Data files for tablespaces containing tables
disk 3	Data files for tablespaces containing indexes

The sections of this chapter discuss each type of data file that you will need to form a complete system. Later, you explore the load balancing and some other concerns that you use when laying out your system. It is important to remember that these considerations are in addition to those of the OFA. You can still use the OFA naming and organization conventions to build the directories that will hold this data.

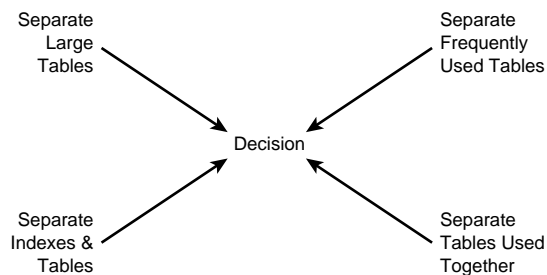
DATA FILES

The most common type of file that you have to deal with is the data file. On large systems, data files consume the vast majority of your disk space. Once, I actually saw a 14G tablespace that consisted of seven 2G disks that stored a single, very large table. These files are also the places where the user data accumulates, so they are the ones that require the closest capacity monitoring. What, therefore, are the concerns that apply to arranging your data files?

First, you typically want to balance the data transfer load between disk drives where you store your objects. You have no control over where a particular table or index is located within a tablespace that has multiple data files. Therefore, if you want to balance the load for several heavily used tables between disk drives, you have to locate them in separate tablespaces whose data files are located on separate disk drives. If you have many disk drives, you may want to distribute the load between the controllers to which the disk drives are attached (see your system administrator). There actually are a number of concerns that you have to balance when you are locating user data objects within tablespaces (see Figure 22.2):

- ♦ You typically want to locate large tables that have heavy queries on different disk drives. This prevents several large queries from impacting one another.
- ♦ The biggest performance gains are often associated with locating the indexes on a separate disk drive from their associated tables. The performance gains come because Oracle makes queries to the indexes and then to the tables for all rows that match the search criteria. Try this before you go in for other fancy reconfiguration efforts when your system is suffering input/output problems.
- ♦ In addition to the size of the tables, you often can get performance gains by understanding which tables are frequently accessed by your applications. Frequently accessed tables include lookup tables with legal values, for example. It is best to spread these tables out on separate disks from your main data disks. In this way, queries for lookup values can proceed in parallel with writes to the basic data tables.
- ♦ Finally, you can also achieve performance gains by understanding the order in which your application accesses data. If you access four large tables in a specific sequence, you may be able to put tables one and three on one disk and tables two and four on another. This prevents the end of the first query or update from impacting the beginning of the next.

Figure 22.2.
Considerations for
arranging user data
objects.



You may also want to consider isolating the effects of one application or group from another when you are laying out your data files. Many instances serve multiple purposes. If you have a single, large data tablespace, you run the risk that one application will use up all the available space and bring down all the other applications that could not access additional disk space. Perhaps you charge the individual users for disk space and therefore need to separate the data cleanly so that you can provide accurate bills. This principle is especially true when you have multiple developer groups sharing an instance. A single developer could fill up all the available disk space on a computer system. Therefore, in addition to quotas on individual developers, it is useful to isolate the developer personal tables from common test tables and other data. Finally, the system tablespace should never be accessible by developers. If you fill up the system tablespace, you can bring down the entire instance and have some delicate work facing you to recover the instance.

Another consideration when you are performing warm backups is that these backups are performed on a tablespace-by-tablespace basis. You can back up the entire instance if you have that much time. However, if you are time-limited, you can back up the tablespaces over a period of time (one week, for example). This is an especially attractive option for large data warehouses. For the warm backup to be effective, you have to get all the data files associated with a given tablespace during the course of the single warm backup. You cannot back up half the data files in a tablespace one night and then catch the rest on the following night. This could lead, in some instances, to splitting objects between tablespaces (and therefore data files) to enable you to have chunks that can reasonably be backed up during a given backup run.

Oracle 7.1 presents a new, interesting consideration for laying out your data files. You can designate certain tablespaces as read-only. This works well for tablespaces that are updated only on an infrequent basis (monthly downloads, for example). It can speed up backups because these objects do not have to be backed up during each cycle. Instead, they are backed up only when you receive an update. Therefore, you may have incentive to separate constant lookup tables and other such tables into separate tablespaces, which you can designate as read-only.

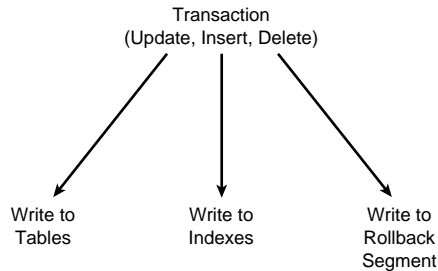
The temporary tablespace presents some interesting considerations. Typically, this tablespace is used to hold data that cannot be sorted or processed in memory during the process of a large query (see Figure 22.3). These queries will be pulling data from the data files that contain the tables of interest and writing it to the temporary tablespace. Therefore, if you perform large queries it is a good idea to locate your temporary tablespace data files on separate disks from those where you store your tables.

Figure 22.3.
Input/output opera-
tions and the temporary
tablespace.



A similar consideration applies to the rollback segment tablespace. Every time you execute a transaction to the database (inserts, updates, and deletes), Oracle makes a record in one of the rollback segments that are located within the RBS tablespace (see Figure 22.4). Oracle also performs a write to a temporary segment in the target tablespace. Therefore, if you have a large amount of transaction activity, it is beneficial to locate your rollback segment data files on disks separately from those of the tables that you typically update.

Figure 22.4.
Input/output opera-
tions and the rollback
tablespace.



Again, on small systems you may not need to perform much in the way of distribution for your data files. Perhaps splitting your indexes and table data files is enough. In some cases, your instance and data query needs may enable you to put everything together on a single disk. There are no absolute rules on how you split your data files. If you at least make a sound basic configuration analysis at the start, you can usually schedule time to move a table or index here and there based on your performance monitoring efforts.

CONTROL FILES

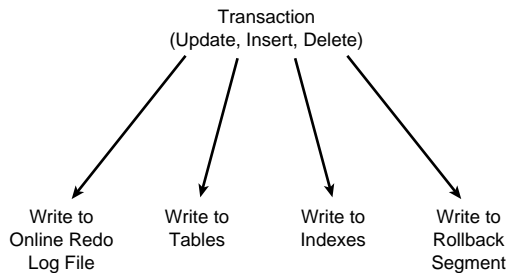
The control files store information that Oracle needs about the configuration of the database and disk files. On all the systems that I have worked with, there were multiple control files that were stored on separate disks. Because the control files are identical to one another, you can recover from the loss of a control file by copying one of the remaining control files to where the missing control file was located. Because they are relatively small (typically on the order of kilobytes), they do not detract from data storage space on whichever drive you choose to locate them.

Because I made such a deal about input/output performance of the data files in the last section, I thought that it would be appropriate to discuss the performance impacts of control files. Basically, control files have little in the way of input/output load. They are updated when you alter the structure of the database (add data files, for example), which is typically an infrequent operation that requires only a small amount of data transfer to the control file. They are also updated on checkpoints, which again are relatively small and infrequent.

ONLINE REDO LOG FILES

Online redo log files can have a significant impact on the performance of an Oracle instance. Therefore, it is worth your time to study the impacts of these files on performance and locate them wisely. Every transaction made to the Oracle database is stored in the online redo log files (see Figure 22.5). Therefore, much like the rollback tablespace data files, you should try to locate the redo log files on disks that do not contain tables and indexes that are subject to heavy transaction load. You also can improve performance by not locating the redo log files on the same disks as those used for rollback data files.

Figure 22.5.
Redo log files and
transactions.



Another important consideration for Oracle 7 databases is the capability of having multiple groups of mirrored online redo log files. This provides you with two benefits. First, it provides redundancy so that if you lose a disk drive containing log files and there is at least one member of each of the log file groups on another disk drive, your instance can continue processing transactions. That alone makes it worth considering for instances that have high availability requirements. The penalty for this, of course, is that you now have to perform multiple log file writer operations for every transaction (one for each member of the group).

The final consideration regarding online redo log files is the size of the individual log files. You have several competing factors. Oracle recommends keeping the log files small to prevent large intervals between checkpoints that would delay recovery. It also recommends looking at your alert log during operations to ensure that you are not wasting time waiting for online redo logs to complete their archiving process. If

you see that you are frequently waiting, Oracle recommends adding more log file groups. There is another alternative to consider. There are some bugs in the Oracle 7.1 software that occasionally cause the Oracle instance to crash when it is switching log files (you get an ORA-0600 error message on large updates at seemingly random intervals). The best solution to this problem is to create a slightly larger log file. This minimizes the number of log switches that have to occur and seems to cure these crashes (this bug is supposed to be fixed in version 7.2). In data warehouses where you have very large nightly updates, you may also want to consider slightly larger log files to minimize the number of files that you might have to restore in the event of a disk loss. For many purposes, the size and number of online redo log files that are created by default during database creation are adequate.

ARCHIVE LOG FILES

For those who may have forgotten my long-winded discussions about the problems that you can get into if you run out of space for archive logs, here is a brief summary. If you fill up your space for archive logs (be it tape or disk space), you will be unable to accept any more transactions that would overwrite an unarchived online redo log file. The Oracle instance will stop up and you may have to shut it down to get it started again. Therefore, ensure that there is a reasonable amount of space allocated for your archive logs.

Once you have secured enough space for the archive log files, it is time to look at where you want to place these files. Archive log files are written to in a mass transfer whenever an online redo log fills up and a log switch occurs. Therefore, you may want to locate the archive logs on a disk separate from the online redo log files. Again, in small instances this does not make a significant difference. However, if you have large data warehouses that have extremely large transfer volumes during their batch updates, your performance can increase noticeably if you do not waste time waiting for a log file to be archived.

THE CONFIGURATION PROCESS

So far, you have explored a number of considerations for arranging the various files that you will need in your Oracle database. You will be given a limited number of disk drives and you may not be able to allocate a separate disk drive for each type of file. How then would you decide which files to assign to various disks? The actual answer varies according to your individual situation. However, never being shy about offering opinions, I would suggest that you consider the following when deciding where to allocate files with a limited number of disks:

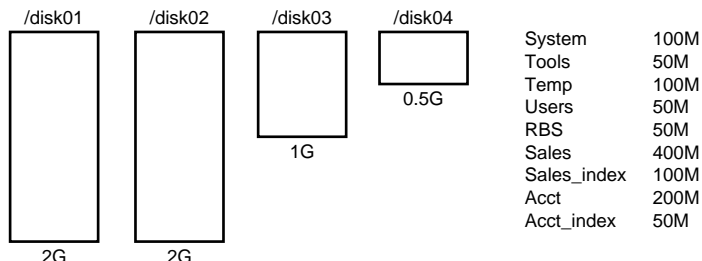
- ◆ In any instance that relies upon indexed tables, try to split the tables and their associated indexes onto separate disks. Whether you are reading or writing, both the table and the index will be accessed on a transaction.
- ◆ If you are heavily transaction-based and relatively light on complex queries, consider splitting rollback segments and redo log files on separate disks from the user data files first. If you have to compromise, it will probably hurt you less to place the temporary tablespace on a disk with the data files.
- ◆ If you process a large amount of complex queries that perform sorts, consider splitting the temporary tablespace from the user data files first. Sorts to disk are slow enough as is (memory-based sorts are orders of magnitude faster) without adding disk input/output contention to the picture.
- ◆ Consider archiving directly to tape if space is really tight. You need to ensure that your tapes can handle the volume of transactions that are expected during the normal workday. However, you can usually perform your system backups at night and then leave a tape in there for the following business day.

The typical configuration process starts with the amount and size of the disks that are allocated to the Oracle database. The first thing that I like to allocate is space for two copies of the Oracle software (the current release and the next release). To me, this is the easiest “given” in the system, and it can be located on disks that serve other purposes because this area is typically not accessed very often. I guess that I always look on this as getting the easy one out of the way first.

The next thing that I try to do is place some of the smaller files on disks that have only small amounts of space available on them. Perhaps you have just a small disk partition on a system disk left over that is not large enough to contain any of your large data files. This could be an ideal location to put one of your control files or tools tablespace data files. These are all easy-to-place, smaller pieces of the puzzle, and I like to just get them out of the way.

A series of data and log files that have to be mapped to the available disk drives are left. I usually like to draw a representation of the disk space available similar to Figure 22.6 on a marker board or sheet of paper. I then list the data and log files and the required space for them down the side. After that, I just try different configurations until I find one that seems to best meet the needs and accommodate all the data. There is no magic formula; just keep trying until you get a combination that seems to work best. Experience with jigsaw puzzles can come in handy here.

Figure 22.6.
Fitting required files
into available disk
space.



A consideration that you may have on some systems is that different storage devices and data transfer busses may provide higher data transfer rates. Check with your system administrator or hardware experts as to whether all the drives have similar speed characteristics. Another common example of performance differences occurs in systems that have both regular SCSI controllers and some of the new fast/wide SCSI controllers (which have approximately twice the data transfer bus bandwidth).

After you have mapped the data and log files to your available disk drives, you still have a few tasks left. First, you have to assign names to the drives and map them to file systems. You should create directories under these file systems to hold data for various instances according to the OFA architecture.

EXPANSION OF THE DATABASE

Because nothing remains constant, it is a good idea to figure out how you will be expanding your database before you even create it. Specifically, it is good to know which tablespaces will tend to expand. If there is extra disk space available on the system, you should map out what new data files are going to be required for the tablespaces that will be expanding. Your goal is first to have a plan. You always look good when someone comes in with a crisis request and you are ready for it. However, there is also the practical consideration that you will want to ensure that the new data files you will be creating will not be on a disk that will have input/output conflicts when these files are created. You do not want to have to start rearranging files before you expand your disk space. The expansion list might look something like this:

Disk 1	500 MB free	Temporary tablespace
		Rollback tablespace
		System tablespace
		Log files
Disk 2	200 MB free	Tablespaces for tables
Disk 3	700 MB free	Tablespaces for indexes
New disks		Expansion beyond above limits

SUMMARY

This chapter has shown you some of the things that you need to consider when you plan out where you are going to put your data and log files. There may be additional local considerations that depend on your particular computer system or company policies. Some organizations have rules against mixing software and data on a single disk drive, for example. Others may wish to keep all the data on external disk drives that can be moved between systems in the event of a problem. The analogy of fitting the pieces in a jigsaw puzzle is a good one in this case. The goal is to have enough disk space so that you do not have to play too many games to make all the pieces fit.

-
- This is a detailed black and white illustration of a celestial globe or map. The top left corner depicts a landscape with a waterfall and a city. The right side features a circular zodiac chart with signs like Gemini and Cancer. The bottom left has the text "MAR" and "THE TEMPERATE". The bottom right shows a map of the Azores and other islands.

A Routine Maintenance Schedule

Perhaps it is a throwback to my days in the Navy when they had formal schedules worked out for each piece of equipment that told how and when it was to receive maintenance. Perhaps I just worry that, with everything that goes on in a typical work week, I will forget to do some of those planning or checking tasks. Whatever the reason, I just feel more comfortable when I take a little time to lay out a schedule of tasks for myself and then keep track to ensure that I get things done. I realize that not everyone is as fond of making schedules; however, this may be beneficial for new DBAs. You also may want to think through some of the processes that I go through here even if you do not want to write your plans down on paper.

So far, this book has presented a number of tasks that the DBA is typically asked to perform. Later chapters present additional tasks that you have to juggle among the emergency requests, hardware problems, and requirements meetings. This chapter takes you through the process of making up a maintenance schedule to help keep up with these routine tasks. There are some things that you need to consider that may conflict with or need to be coordinated with common DBA tasks. This chapter also presents some sample schedules. They may not fit exactly into your environment, but you can modify them to fit rather than start from scratch each time.

OVERVIEW

For purposes of this discussion, the myriad of tasks that you may be asked to perform are divided into two categories—planned and unplanned. This is not an exact science and you gain benefits if you can organize the majority of your tasks and then handle the others as they arise. Some tasks occur on both a planned and unplanned basis and are placed in both categories. The emphasis here is to focus on what you can put in your work plan on a controlled basis—the planned tasks.

When you start your task list, merely write down the items on a sheet of paper. Do not try to qualify any of them or categorize them. Just write them down. In this phase, you are just brainstorming. Do not be concerned yet about how you will implement these task requirements. A basic list might include the following:

- ♦ Routine backups
- ♦ Copying archive log files to tape during the day, if needed
- ♦ Routine checks of space utilization and free space in tablespaces
- ♦ Routine checks of fragmentation of tables
- ♦ Routine checks of security privileges—object and system access
- ♦ Routine checks of the database configuration
- ♦ Routine checks of the database tuning
- ♦ Checks during the day to ensure that the database is still operational

- ◆ Routine checks for users who no longer need access to the database
- ◆ Routine checks for tables and other objects that are no longer needed
- ◆ Routine checks of log and trace files to see whether there are any problems requiring action; routinely cleaning out old entries from the log files

There could be many additional tasks that you can perform from a technical standpoint. The ones listed are a compromise between what is feasible to accomplish and where the common problems with systems occur. You should also consider a list of additional tasks that may be more focused toward meeting local, nontechnical requirements. I cannot guess what your local organization climate will require, but I'll throw out the following to stimulate your thinking process:

- ◆ Weekly or monthly status reports
- ◆ Database security access reports
- ◆ Regular status meetings

You may wonder why I brought up the last set of tasks in this discussion. It was to show you that you need to brainstorm all the things that will eat up your time. In the next several pages, you explore some of the other activities that you need to consider including. Conflicts are listed and ways that you can make two tasks work together are presented. For example, it may be useful to schedule your routine space utilization analysis to occur before your regular status meetings. That way, you will have answers ready to the questions that people are likely to ask.

Once you complete your brainstorming of tasks that could eat up your time as a DBA, it is time to start qualifying them. Remember, in the previous exercise, you just listed an idea as it came into your head. You did not try to evaluate whether you should do it. Some of you with a very practical disposition may have had trouble with the free-thinking brainstorming exercise. However, now you will have your chance to evaluate the list of tasks that you created for practicality. Consider the following list of criteria when judging the practicality of a given task for your situation:

- ◆ How much of your time will it take to complete this task? Remember, your time is both valuable and limited.
- ◆ What are the benefits of this task to your organization? Every organization has a different set of priorities. Some places are extremely sensitive about security, for example. Other groups may provide complete access to all data to every member of their staff. You have to decide which are the hot buttons in your local world.
- ◆ Is it a management requirement that you are not going to be able to avoid? Sometimes it is easier to mark the ones that you are going to have to do whether you like it or not and then move on to decide on the other tasks.

- ◆ What problems have burned you in the past? Different applications tend to suffer different problems. Data warehouses tend to be extremely space-sensitive. Instances where the developers frequently make their own tables tend to suffer from fragmentation problems.
- ◆ What problems have you inherited from the existing culture? Perhaps your data center manager was burned 20 years ago when a disk crashed and the backups were bad. In such organizations, backups may take on the aura of a religion. My experience shows that this is very hard to change, so it is usually best to adapt to it and devote extra time to the backup process.
- ◆ What are the strengths and weaknesses in the organization and the system? If there are no operators for the system, you may wish to devote a little extra energy to checking to see that the system is operational during the day (before a user has to call and tell you).
- ◆ Which activities are not your problem? In some databases, there is a separate security administrator who is responsible for all the security monitoring and new account generation. Some database backups also are performed by the system administrator and not the DBA. This is another example of making your list simpler by crossing out items that you do not have to worry about (you probably have more than enough to worry about as is).
- ◆ Can the task be automated? In the next chapter, you learn some techniques to enable the computer perform the tasks for you and then just send you a report when it is done. Your schedule may be full, but computers often have some free time in their daily schedules.

The next step in the process is to come up with a desired frequency for the tasks that have made it past the screening process. Some checks, such as security reports, do not need to be run every hour. Others, such as checking whether the instance is still running, are useless if run only once per month.

At this point, you should not be nervous. Just make your best call as to the way you feel the tasks should be performed. This may be tough if you are new to the DBA and computer support roles (for example, you are an electrical engineer who got stuck with the job). Many plans are iterative in nature. If you find that you run into problems regularly in one area (such as tablespaces filling up), increase the frequency of your monitoring in that area. Just having some kind of plan, even if it only sits in your desk drawer, puts you ahead of most of the other DBAs out there in terms of planning and organization.

So far, you listed the tasks that you will regularly be called upon to perform, figured the value versus the time cost, and took a shot at listing how often you want to perform these tasks. Remember, this plan evolves over time as you gain experience. You do not have to get it perfect on the first try.

The next step is to develop a schedule as to when each of these tasks will be performed (the first of the month, on Tuesdays, and so on). However, before you can do this, there are a few other things that you need to consider. In the next section, you factor in the user and system processing activity around which you have to work. Most DBAs do not have the luxury of taking the database down in the middle of the day to perform a cold backup. Figure 23.1 shows what your list might look like so far.

Figure 23.1.

Sample list of tasks
with suggested frequen-
cies.

<u>Task</u>	<u>Frequency</u>
Complete, cold backup	M–F at night
Tuning report	weekly
Security report Not Needed	_____
Configuration report	monthly
Tablespace utilization report	daily
Log file monitoring & clean out	weekly
Check database operational	hourly

STARTING WITH USER AND SYSTEM PROCESSING SCHEDULES

In an ideal world, you are the absolute master of your own destiny. The world revolves around you and you set all the schedules based on your convenience. However, because a DBA is usually a support role and the main purpose of the system is to provide a service to the users, the users' schedules are usually the starting point for your activities. Not only do you have to work around the schedules of your users, you also have to coordinate your activities with those of other support groups, such as the system administrators and network administrators.

The first thing to start with is a list of user processing requirements. In most cases, there are a number of user processing cycles beyond the daily scheduled hours of availability. You will have processing peaks at certain times of the month, quarterly, yearly, or perhaps at other intervals. Some of these may impact the schedule that you are preparing and some may not. Again, I recommend that you start brainstorming all your expected user processing demands by listing them on a sheet of paper. Here are some suggestions to get your creative juices flowing:

- ◆ The published hours of availability for the users of the system. This is why the system was purchased.
- ◆ Any periods where developers perform time-consuming tasks such as compiles or back processing.
- ◆ Any periods of special processing or report production. Examples might include monthly, quarterly, and annual closing of the books for financial systems.

- ◆ Times scheduled by system administrators, network administrators, and other support staff for their activities, such as system backups and network maintenance. You can often negotiate with these folks if it makes sense to move things around.
- ◆ Hours of access to the building or computer room.
- ◆ Hours where operator support is available.

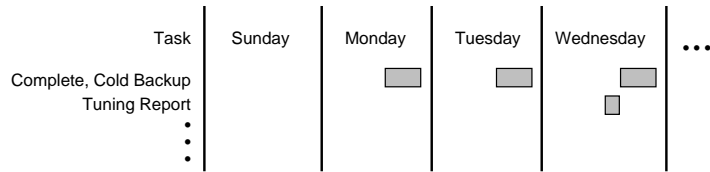
As with so many issues in this book, you have to take a look at your own system and local rules to determine which items you have to add to your list. So far, you have added things that might have some form of impact on your DBA schedule. Do not try to think through all the ways you can work around a particular user requirement to perform your processing. For now, just put these tasks down on the list along with when they occur in the processing cycle.

There are a few points to consider when making up this list. If you support local development, you may also have to factor project schedules into your calculations. The project tasks may not have a regular interval during which they happen, but you should at least list what you do know. Examples of development project events that might interfere with some of your work include weekends where the developers are running integrated system tests or nights when they plan on recompiling all their applications. Although they may not like to commit to an absolute schedule far in advance, you can usually get a dialog going with them to ensure that your activities do not conflict with theirs. With most development projects, you should schedule some leeway around their promised dates in case unexpected events force them to run a little longer than they planned.

You now have a list of all known scheduled events that might impact your schedule making efforts. Associated with each of these events, you will have a scheduled time. This scheduled time may be a fixed period such as January 4–17. It may also be a recurring time—the first two days of every quarter are devoted to closing the previous quarter's books, for example. If you have worked at a particular location for a long time, this may seem like second nature to you. However, it is a good idea to put this down on paper so that you can see the overview when you are scheduling your new activities as a DBA.

Next, it is a good idea to draw out a calendar of the recurring tasks so that you can visually see what is happening. In project scheduling terms, the format that I prefer is referred to as a Gantt chart. The key to this format is to list all the tasks that need to be performed down the left side of the chart and draw a time line across the top of the chart. You then draw boxes for each task under the appropriate times that you will be performing these tasks. Figure 23.2 illustrates the basics of this chart.

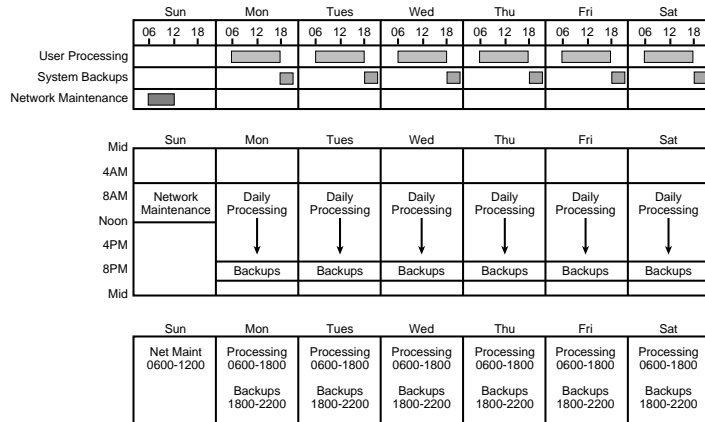
Figure 23.2.
Basics of the Gantt
chart scheduling
technique.



At this point, you are not quite ready to complete your routine maintenance schedule. You have listed the fixed periods that you may have to work around, such as hours of system availability and times that other support staff will be performing work that may impact your system. Now you need an analysis of what activities conflict with one another. That is the topic of the next section. However, for now, it may be useful to start your schedule charts by filling in the fixed events that you have documented in this section.

You have a number of options as to how your schedule will look. You can draw it on a weekly basis, a monthly basis, or even a daily basis. It depends on how fine you want to schedule your time and what the common intervals for your tasks are. I like to make a guess at what I think will work and jump right into the scheduling process. I use a pencil or do it on the computer so that I have plenty of opportunity to change tasks and move time periods around. This is a skill that is best learned by doing, so give it a try. Figure 23.3 illustrates a set of sample schedule charts that show fixed user and system processing schedules drawn in a variety of formats.

Figure 23.3.
Fixed user and system
processing in different
formats.



TYPES OF ACTIVITIES

A big buzzword in the computer industry today is *parallel processing*. It is a nice concept because it has a number of different tasks running in parallel that efficiently use computer resources. Unfortunately, although a number of users can run reports

in parallel if you have adequate processing capacity, many of the tasks that you perform as a DBA conflict with other activities. Therefore, your goal is to look at each of the tasks that you have laid out for yourself and determine what you need to have or need to avoid in order to get the job done.

Before you start on the really technical considerations, let's look at an organizational culture issue. Many organizations have been doing computer tasks in a certain fashion since before many of you were born. It is true that some of these tasks could be done in a different order or at different times. However, unless there is a compelling technical reason to suggest a change, it may be to your benefit just to accept the given cycle of activities if they will get the job done. For example, many mainframe systems perform their backups somewhere in the wee hours of the morning after the nightly batch cycle has been completed. There is probably a UNIX system in this data center that does not have a nightly batch need and you could back it up any time from 6:00 p.m. to 6:00 a.m. You may have the bright idea of scheduling your backups in the early evening so that they level the load of tape mounts and avoid getting telephone calls in the middle of the night if there is a problem with the backups. However, it's a good idea to talk to your operations managers to see whether they would rather stick to the traditional schedule. It may be easier to explain to new operators; it also may be easier for the shift supervisor to manage.

When looking for conflicts, start by determining when your computer system and network is not available. If the system administrator wants to perform a complete system backup with all processes down (including your Oracle instance processes), you can mark this off on your schedule as dead time for most of your tasks. You may also find times when the network is not available that may limit some of your activities. Another typical conflict is the time during which you don't have operator support available. Finally, you should consider resource conflicts. An example of a resource conflict is when both you and the system administrator want to use the only tape drive that is installed on a particular system.

There are some tasks that could theoretically be done with the system in use, but that you may consider doing at other times just for safety. For example, the hot backup process could function during periods of heavy transaction activity. However, Oracle recommends that you avoid these periods when performing hot backups because you can run into problems if you have a system crash in the middle of the backup. Some of your tasks may demand a large amount of CPU or input/output capacity. You may wish to be considerate of the other users and schedule this for off hours to avoid slowing the system down excessively.

You will also run across a set of DBA tasks that need to be done alone. Major reorganizations of tables within tablespaces and de-fragmenting a tablespace need to have a quiet database that is not receiving transactions (see Chapters 27 and 35 for more information). Schedule any activity that is serious enough to require a

backup beforehand (software upgrades, major reorganization, and so forth) when you have the system to yourself. It may be a challenge to schedule such time on some systems, but the risks of losing user data or transactions in process are high.

Other activities that are easy to work in are those such as running the monitoring reports that will be discussed in Chapter 30. These items have little impact on anything else and can pretty much be run any time the Oracle instance is up. The only requirement for the tuning reports is that the statistics gathering tools internal to Oracle are reset at startup. Therefore, you need sufficient time since the last instance startup to ensure that your statistics are valid. The activities that fit into this category should be left until last because they can fill any convenient gaps.

Chapter 24 discusses some techniques to automate some of the routine DBA tasks. This can be important because often the only time that many activities can be performed lies outside the normal working day. However, if you long for a life outside your office, scheduling automated tasks to run in the middle of the night that leave a report waiting for you in the morning can be very helpful.

This section has focused on the possible conflicts that may come up between tasks. Depending on your exact operating system, backup scheme, and a number of other factors, you may not have many conflicts at all. Conversely, if you have a high availability requirement, you may have to work hard to find time for your necessary DBA support tasks. Figure 23.4 illustrates conflicts in the activities schedule of a DBA.

Figure 23.4.
Sample conflicts in
DBA activities schedule.

<u>Task</u>	<u>Frequency</u>	<u>Conflicts/Needs</u>
Complete, cold backup	M–F at night	Instance shut down
Tuning report	Weekly	Instance operational for a while
Configuration report	Monthly	Instance operational
Tablespace utilization report	Daily	Instance operational
Log file monitoring & clean out	Weekly	Coordinate w/backups
Check database operational	Hourly	Instance operational

THE DAILY SCHEDULE

Recall that you need to figure out which time frame is most appropriate for your own schedules. You also need to figure out which tasks need to be performed and at what interval. The following is the basic process:

1. Brainstorm a list of possible tasks.
2. Evaluate which tasks are worth your effort in your environment.
3. Assign a desired frequency to each of these tasks.
4. Determine the user and system processing schedules with which your database will live.

5. Determine how the tasks interrelate. Some will be conflicts. Some may be logically tied to other tasks.
6. Prepare a basic chart showing the tasks and when they will be performed.

For the sake of argument, I decided to prepare a set of daily schedules for a one-week period (see Figure 23.5). It does not match any one environment that I have worked in, but shows a generic overview of what you might consider as a starting point for your schedules.

Figure 23.5.
Sample daily task schedule.

	Sun	Mon	Tues	Wed	Thu	Fri	Sat
Mid							
2AM		Tablespace util report	Tablespace util report	Tablespace util report	Tablespace util report	Tablespace util report Config Rep (1st week)	
4AM							
6AM							
8AM	Network Maintenance	User Processing	User Processing	User Processing	User Processing	User Processing	User Processing
10AM		↓	↓	↓	↓	↓	↓
Noon							
2PM							
4PM					Tuning Report		
6PM		Backups	Backups	Backups	Backups	Backups	Backups
8PM		↓	↓	↓	↓	↓	↓ Log Clean up
10PM							
Mid							

Remember, this schedule is evolutionary. As you implement it, you will learn how to improve it. New products or applications may require different support and you will adapt your schedule to meet these needs. Finally, it is your tool. I like to publish my schedules so that no one can claim surprise when I take the database down for a backup. This may not work in your environment. Perhaps your schedule will sit on a tablet in your desk drawer for your review only. Whatever form it takes, it has to be a tool that is useful to you and helps you take charge of the tasks assigned to a DBA.

THE LONG-TERM SCHEDULE

Recall that certain items do not occur on a regular basis. Perhaps your company has a lot of reporting to perform during a certain month of the year. If you have development projects, their timelines may influence what support you will need to provide. Perhaps you want to upgrade your database software before developers begin a new project. It may be unwise to de-fragment your development database the weekend before a major application delivery (see Chapters 27 and 35 for more information about de-fragmentation).

In addition to the regular schedules that you developed in the last section, you should also keep track of these other key events that will affect your work. There are a variety of formats that you could use:

- ◆ A wall calendar where you mark the key events in the blocks for the appropriate dates
- ◆ An ASCII text file on the system where you record a date range and the activity associated with it
- ◆ The pages of your personal planner, for those of you who carry such things around (it is helpful for those of us who remember complex passwords and combinations, but forget simple things that we promised to get done)

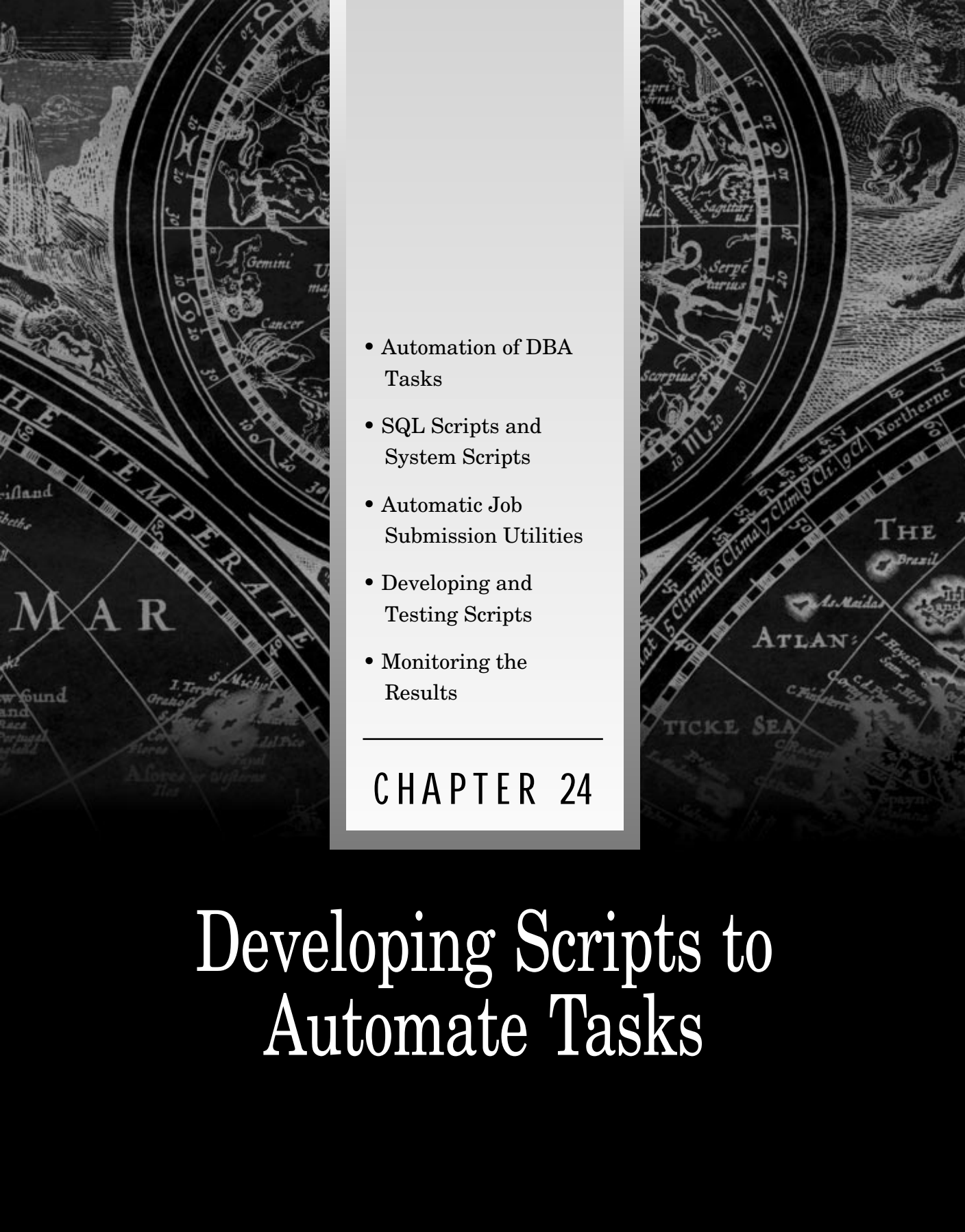
These are useful when you plan your major projects, such as database upgrades or reorganizations. It enables you to check what is going on, both on your regular schedule and your long-term schedule. It is helpful to develop relationships with your system administrators, developers, and other interested parties where you share your project schedules. Then, if you plan a major task of your own, you can let everyone know in advance so that they do not plan to do something that might be a conflict.

SUMMARY

This chapter presented some planning techniques for developing a routine maintenance schedule and a long-term event schedule. Some people will never like to perform this type of planning. Others (like me) cannot face life without some kind of scheduling system. There are a number of authors of productivity books that refer to this type of system as a tool for taking charge of your life. I recommend that you at least consider going through this exercise once to ensure that you have thought out your plans and task list. Then, if you decide that this is all obvious and you can keep it in your head, do so. You'll save a lot of paper and planning time. If, however, you find that you are running in panic mode and you have tablespaces filling up and other problems in a never-ending series of crises, consider a routine maintenance plan. I have implemented such plans and we got to the point where we were almost always able to expand the tablespace before the users' jobs bombed out due to a lack of space. It makes them happier and enables you to do the tasks that need to get done when it is convenient for you. Remember, it is no fun doing emergency database surgery at 2 a.m. on Sunday morning (I know—I've been there).

The next chapter ties in nicely with this one. Here, you worked out a long list of routine maintenance tasks. You developed a schedule that probably included a lot

of work that had to be performed after normal working hours. Some Oracle systems have 24-hour-per-day operator support. In these environments, you can usually leave a series of commands for the operators to type in at the specified times. However, many of your tasks, such as the tuning analyses, are somewhat complex and a bit of a pain to type again and again. In the next chapter you learn the technique of developing scripts (JCL, CLISTs, batch files, or whatever else you call them) to help automate most of the routine jobs that you are asked to perform. Chapter 24 also discusses the basics of using automated job scheduling routines to kick off processing tasks and reports whenever they are needed (when you are at home in your nice, warm bed, for example) and have a report of the results waiting for you in the morning.

- 
- Automation of DBA Tasks
 - SQL Scripts and System Scripts
 - Automatic Job Submission Utilities
 - Developing and Testing Scripts
 - Monitoring the Results

CHAPTER 24

Developing Scripts to Automate Tasks

Chapter 23 described how you could use scripts to automate tasks with special interest. Perhaps you envisioned a world where you would implement a series of these scripts and then sit back at your desk catching up on the mountain of computer magazines that you receive each week. Unfortunately, that is not going to happen. However, a little work spent developing scripts to help automate your tasks can provide you with a number of benefits:

- ♦ *Improved accuracy.* Scripts function the same each time they are run and they are not subject to typing errors once they have been tested correctly.
- ♦ *The ability to run jobs during off hours.* This can take advantage of lulls in the processing cycle.
- ♦ *The ability to run jobs repetitively.* A classic example of this is a script that runs every ten minutes to check whether all the Oracle processes are running and sends you an electronic mail message if there are any problems.
- ♦ *Reduction in typing.* Some tasks, such as the tuning checks, require a fair amount of typing. It is much easier to do it once and then proof the script thoroughly. It is even easier if you just copy it from the enclosed disk and avoid typing it.

The basic concept of automating computer administrative tasks has been around for some time. I remember a series of paper tapes that were carefully stored for the first PDP-8 computer that I worked on in high school. They contained boot sequences and diagnostics that were used for both routine restarts of the system and for when problems reared their ugly heads. Of course, back then, the only alternative was to use a switch register to key in binary sequences, so even paper tape seemed wonderful.

Even though support scripts, procedures, and utilities have been around for a while, it may take some time to adapt to the new environment. This is more of a challenge when you are changing your operating system in addition to changing your database management system. You *can* get the job done in any environment once you learn its syntax. In the Oracle world, you interact with the database through the Structured Query Language (SQL). If you want to use programming constructs such as if-then statements or program loops, you use the PL/SQL programmatic extension. Finally, if you want to start your commands from the operating system prompt or build more complex routines, you have to learn the shell scripting language for your operating system. Shell script is a term from the UNIX world, but VMS has the DCL, and almost all the operating systems that I have run across have some kind of scripting language.

What kind of scripts would I typically use as an Oracle DBA? My list of favorites includes the following:

- ◆ Routine monitoring reports (utilization, tuning, security, and configuration)
- ◆ Scripts to add new users to the database
- ◆ Tools to perform routine data uploads
- ◆ Utilities to update summary data tables from base data tables (data warehouse types of instances)
- ◆ Any other task that is routinely performed and requires substantial typing of commands

This may seem to be just one more thing that you are being asked to learn when you are already overloaded. That's a fair complaint in many cases wherein people who are already overworked are asked to pick up new technology. The good news is that you do not have to learn it all at once. This chapter provides a general introduction to some of the script programming concepts that I have put to use in my DBA assignments. The disk included with this book has a number of SQL scripts and UNIX Korn shell scripts that are ready to use. You can use these scripts as examples rather than learning from a book on SQL programming. Finally, although learning the basics of scripting seems like a challenge, the goal is to automate some of the tasks that you will need to perform anyway, so it is worth the time.

AUTOMATION OF DBA TASKS

As a DBA, you will be asked to perform a number of tasks. A logical question to ask at this point is which of the tasks that you are going to perform are candidates for automation? Individual preference will determine some of the tasks that you choose to automate; however, the following are good candidates for automation:

- ◆ Tasks in which you frequently type the exact same set of commands, such as instance startup, shutdown, and so forth.
- ◆ Tasks in which there is a large amount of typing, such as issuing the queries used to determine the tuning status of an instance or issuing the commands to take the tablespaces into and then back out of warm backup mode.
- ◆ Tasks in which it is easy to forget optional parts of the command line, such as forgetting to set the default and temporary tablespaces when adding a new user to the system.
- ◆ Any task that you want to run during off hours automatically through a job submission utility such as `cron` under UNIX.

This chapter explores three tools for developing shell scripts to help the DBA—SQL, PL/SQL, and Korn shell scripts. Why these three? First, SQL and PL/SQL come with Oracle. There are no additional purchases and you can pretty much guarantee that most installations will have them installed (PL/SQL is called the procedural option in some versions of Oracle). I chose the Korn shell script because it is one of the most popular and easiest to learn of the UNIX shell scripts. I chose a UNIX tool because that is where Oracle sells most of its databases. However, I have implemented many of the same types of concepts under VAX DCL and believe that you can adapt them to almost all modern computer environments.

These are three relatively simple tools. They do not have fancy compilers, debuggers, or graphical interfaces. They are the brute force approach to programming, but they have the blessing of being based on standards that you will see in almost all UNIX Oracle environments. Each of these tools fills a different role:

- ♦ SQL provides the basic interaction with the database and instance. They are executed from an Oracle product such as SQL*Plus or SQL*DBA.
- ♦ PL/SQL goes around the SQL statements to provide programmatic constructs such as if-then and while loops. You get greater speed using pure SQL, but sometimes you need to use a loop of checking to see whether a condition is true before you can proceed. These are also executed for an Oracle product such as SQL*Plus.
- ♦ Korn shell scripts provide control of processing at the operating system level. Automatic job submission utilities such as `cron` run UNIX shell commands. You can also build shell scripts and put them in a directory that is in your path so that you can execute routine jobs from wherever you are working.

The first sample is a very simple SQL script, `stopora.sql`, which is designed to work from within SQL*DBA. Its purpose is to shut down an Oracle instance. The command script reads as follows:

```
connect internal;  
shutdown immediate;  
exit;
```

This script will get the basic job done for you. However, you have to remember the directory name that you put this script under each time you want to execute it. Because I tend to work with a fairly large number of directories, I prefer not to have to remember such things. Instead, I build a Korn shell where I type the details once and then just execute the shell script. For example, if you create a script called `stopora` and place it in a directory that is in your search path, you can type `stopora` from the UNIX command prompt anywhere on the system to stop the instance. The `stopora` file contains a line similar to the following:

```
sql> sql> lmode=y < /users/jgreene/bin/stopora.sql
```


You can have a single shell script call a number of SQL scripts. You also can mix in other shell script commands, such as `date` (which prints the date and time for future reference). If that is not complex enough for you, you can have SQL and PL/SQL scripts call other SQL and PL/SQL scripts. I typically construct a Korn shell script to run a series of SQL scripts directly. It tends to make it easier for me to maintain when I am not dealing with layer after layer of programming. However, you should do what you are most comfortable with when it comes to programming.

There are products on the market that perform many of the functions that you want to automate. Products such as Oracle Server Manager and products from third-party vendors provide easy GUI interfaces and script-like capabilities. However, very few DBAs have access to such tools, so you will learn the old-fashioned way of doing these tasks in this chapter. It can't hurt to learn a little about scripting, because you can always think up a new function that is not provided by your DBA support tools.

SQL SCRIPTS AND SYSTEM SCRIPTS

This section explores some of the basics of the scripts that you will typically construct as a DBA. Full-time programmers who are writing your user applications will use the full capabilities of SQL, PL/SQL, and shell script languages. DBAs, however, typically need to learn only a few tricks to automate basic tasks that they need. Let's go over a few of those tricks.

The first category of script is the simple SQL command list. This is merely a text file that contains the list of SQL commands that you type at the command prompt. The shutdown script example in the previous section is an example of this type of script. A few points are in order regarding this type of script:

- ◆ End each of the SQL commands with a semicolon (;). This is the end-of-line character recognized by SQL*DBA and SQL*Plus. The forward slash (/) can also be used to cause SQL commands to be executed within the script.
- ◆ These commands are executed against whatever Oracle instance you have set up in your environment when you execute this script. If you want this script to run only against a specific instance, you need to put commands in a shell script file that calls this SQL script.

The simple SQL command list is probably the easiest script to deal with when you are first beginning. There is very little in the way of syntax other than the rules that you need to learn to issue the commands manually from the SQL prompt. Reports written in SQL*Plus are an extension of the basic SQL command script. Most of the reports on the enclosed disk were written in SQL*Plus. SQL*Plus is designed to be a command-line interpreter, and there are many utilities that are better at the job of preparing reports. However, all DBAs get SQL*Plus with their systems, so it has the blessing of being there.

SQL*Plus is a SQL interpreter that also has a series of formatting and control constructs that you can use to obtain adequate output from the system. Specifically, the following commands are frequently used to prepare reports using SQL*Plus:

- ◆ The prompt command enables you to send text to the screen. It can be used to display instructions or comments on your reports. Although SQL*Plus has functions that handle headings, it is often easier just to use the prompt command to put some text at the top of your report.
- ◆ The column *column_name* format *format_string* command enables you to control how wide the output for a given column in a query will be when it is displayed. It also enables you to insert commas and dollar signs in numbers. The SQL*Plus references show you the details of what you can do to adjust your formats with this command. Here are some sample format commands that you may use:


```
column anumber format 999,999,999
column sometext format a20
```
- ◆ The spool command enables you to redirect the output of SQL*Plus to go to a text file or printer in addition to being displayed on the screen. If you just type spool, your output goes to the default printer that you have set up. If you type spool followed by the name of a print queue, it is sent to that printer. Finally, if you type spool followed by a filename, the output is placed in that file in ASCII text format.

Most of the rest of the SQL*Plus reports are composed of normal SQL statements that retrieve data from various tables in the system. The following is part of the DBA tuning report discussed in more detail in Chapter 30:

```
rem Library Cache checks

column libcache format 99.99 heading 'Library Cache Miss Ratio (%)'

prompt Library Cache Check
prompt
prompt Goal:  <1%
prompt
prompt Corrective Action:  Increase shared_pool_size
prompt .                  Write identical SQL statements
prompt

select sum(reloads)/sum(pins) *100 libcache
from v$librarycache
/

prompt
prompt

rem *****

rem Data dictionary cache checks
```

```
column ddcache format 99.99 heading 'Data Dictionary Cache Miss Ratio (%)'

prompt #####
prompt
prompt Data Dictionary Cache Check
prompt
prompt Goal: <10%
prompt
prompt Corrective Actions: Increase shared_pool_size
prompt

select sum(getmisses)/sum(gets) * 100 ddcache
from v$rowcache
```

This works well when you want to get information from the Oracle database. A common desire is to set up a script that performs some kind of action on the database. The only problem is that you want to change a few of the data items in the command. SQL*Plus provides the `accept` command to enable you to ask the user for input; then you can use the value input in later statements. For example, suppose you want to build a script to add a user to your database. You will always assign that user to the `developer` role, set the default tablespace to `users` and the temporary tablespace to `temp`. The following script enables you to ask for the user ID and then add the user to the system:

```
accept newid char prompt 'Enter the new user ID : ';

create user &newid identified by &newid
default tablespace users
temporary tablespace temp;

grant developer to &newid;
```

PL/SQL provides a number of programming constructs that you may wish to use at some point. It enables you to use variables to calculate and store values. It provides you with a loop that can execute a series of instructions a certain number of times or until a criteria is met. Finally, it enables you to make decisions on when to execute certain steps. Although this is an interesting topic and a subject of entire books, it is probably not needed for the basic tasks of routine automation of DBA tasks. Here is an example:

```
-- *****
--
-- Script:      data_xfr.sql
-- Purpose:    Transfer data between the golf_scores table and the
--             golf_averages summary table
--
-- Revisions:
--
-- 4/5/95      Script created (Joe Greene)
--
-- *****

declare
```

```
    acounter      number(3);

begin

    insert into golf_averages
        select avg(score),name
        from golf_scores
        group by name;

    acounter := acounter+1;

end;
```

SQL scripts and PL/SQL blocks enable you to do many of the tasks that you need to perform. However, there are some limitations as to what you can do within these utilities, which were designed for database interaction. First, these scripts run only within SQL*Plus or SQL*DBA. They cannot be executed directly from the command line. Second, there is no provision to configure which instance you are attached to within these scripts unless you have SQL*Net and can reconnect using the `connect user_ID/password@connect_string` format. Finally, these scripts cannot access the operating system utilities that you may wish to use, such as electronic mail. That is where shell scripts come into play. Shell scripts consist of commands that you can type at the operating system prompt to execute jobs.

There are three components in the SQL*Plus command line that are needed to execute a script. The first is the command that the operating system recognizes as a call to the SQL*Plus binary. This is the word `sqlplus` (in lowercase letters). The next component is a valid Oracle user ID and password. You separate these two components with a forward slash (/). Finally, you need to tell Oracle the name of the script that you wish to execute. This component of your command line is preceded with an ampersand (@). If you are currently in the directory that contains the script name, you can merely type the script name. You can also type the fully qualified name of the script (which includes the full directory path). The `.sql` extension at the end of the script name is optional. Some examples of calls to SQL*Plus scripts from the command line include the following:

```
$ sqlplus system/manager @/users/jgreene/tunecheck.sql

$ sqlplus bsmith/sarah @utilcheck
```

Many functions, such as startup and shutdown, need to be run using SQL*DBA. It can also be convenient to use SQL*DBA because you can run DBA scripts using the `connect internal` option. In this case, you do not have to supply a user ID and password on the command line as in SQL*Plus. Instead, you need to ensure that your first line in the SQL script contains `connect internal;`. SQL*DBA's output formatting options are somewhat limited, but it is otherwise functional. To call a script using SQL*DBA, you typically need three components on the command line. The first part is the call to SQL*DBA, which is `sqldba` (again, lowercase letters). The

next component is the command `lmode=y`. This instructs SQL*DBA to come up in the line interpreter mode as opposed to the menu-driven interface. Finally, you need to tell SQL*DBA the name of the script to run. You indicate this by using the UNIX redirection operator for input, which is the less-than symbol (`<`). Here are some examples of calls to SQL*DBA:

```
$ sqldb a lmode=y <testscript
```

```
$ sqldb a lmode=y </users/jgreene/sql/testscript.sql
```

Another important technique in shell script programming is setting up the environmental variables desired for a particular job. This becomes especially important when running jobs through automated job scheduling utilities rather than from the command line. You do not have the opportunity of logging in and setting up the correct environment before running jobs through UNIX's `cron` or `at` utilities. This is a relatively simple two-step process in the Korn shell. You set a variable equal to the desired value and then export that variable. To access the contents of the variable within your script (for example, to print it), you prefix the variable with a dollar sign (`$`). Here are some examples:

```
$ ORACLE_HOME=/usr2/oracle/product/7.1.4
```

```
$ export ORACLE_HOME
```

```
$ ORACLE_SID=test
```

```
$ export ORACLE_SID
```

The Korn shell provides basic text display and variable input capabilities for your needs. The input is accomplished using the `read` command. The basic format is the word `read` followed by a variable name. You do not need to initialize shell variables before you use them. The output is accomplished by the word `print` followed by either a string of text enclosed in double quotation marks (`"`) or the dollar sign (`$`) followed by a variable name. There are a number of format control items that you can put within your `print` statements. If you type `-n` after the word `print`, it will stay on the same line after it is finished displaying your output. If you place `\n` inside the quotation marks, it skips an extra line. The following are some examples of input or output operations:

```
print -n "Input the new user ID: "
read newuser
```

```
print "\n\nOracle Instance Tuning report"    # Two extra lines skipped
```

```
print -n "New user ID entered was "
print $newuser
```

There is much more that can be learned about SQL and shell script programming. For example, loops and decision statements under the shell script can enable you to write some fairly sophisticated programs. There are a wide range of programming constructs and a number of useful functions within the SQL and PL/SQL world that can provide services that you might need. However, you now have a level of

knowledge that will enable you to get started on building the scripts that will help you as a DBA. There are number of scripts on the disk that you can also use to learn some of the tricks.

Let's look at a sample Korn script, used to add a batch of users to an instance. It prompts you to define a role for the users (in this instance, all privileges are inherited from roles) and also remembers to assign the correct tablespaces (Oracle tends to default to the system tablespace for users). It includes Korn shell if-then and while loops, but if you have used these constructs in other programming languages, you should be able to pick up the ideas fairly quickly even though the syntax may be slightly different. This script illustrates what is possible in this type of programming. It is a Korn shell script that gets some input from the user, builds a SQL script based on the user's input, and then executes that script;

```
# *****
#
# Script:      adduser
# Purpose:     Script to add users to the database in a controlled manner
# Revised:    10/5/94
#
# *****

default_ts=users
temp_ts=temp

# *****
# *****
# You should not have to modify below this line when porting the script
# *****
# *****

# Clear screen and get data from user (who must be in the DBA group, not just
# have DBA privileges within Oracle

function get_input {

    print "connect internal;" > adduser.sql

    while [[ $user != "exit" ]]; do

        clear

        print "New User Addition Script"
        print "===== "

        print "\n\n"
        print -n "Oracle user ID (exit when done): "
        read user

        if [[ $user != "exit" ]]; then

            print "\n"
            print -n "Role to be granted: "
            read role
```

```

print -n "create user " >> adduser.sql
print -n $user >> adduser.sql
print -n " identified by " >> adduser.sql
print $user >> adduser.sql >> adduser.sql
print -n "default tablespace " >> adduser.sql
print $default_ts >> adduser.sql
print -n "temporary tablespace " >> adduser.sql
print -n $temp_ts >> adduser.sql
print ";" >> adduser.sql

print -n "grant " >> adduser.sql
print -n $role >> adduser.sql
print -n " to " >> adduser.sql
print -n $user >> adduser.sql
print ";" >> adduser.sql

fi

done
}

# *****

# Get input and then run the script created

clear

. oraenv

get_input

chmod 700 adduser.sql

clear

print "Now to run the script to perform the additions"

sqldba lmode=y <adduser.sql

rm adduser.sql

# *****
# END OF SCRIPT
# *****

```

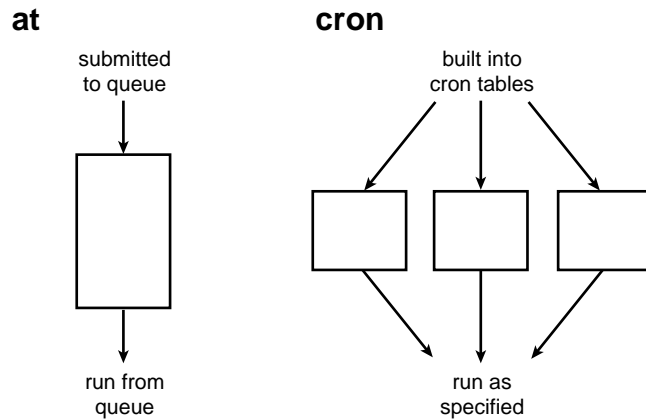
AUTOMATIC JOB SUBMISSION UTILITIES

Automated job submission is a concept that has been around for a long time. Because computers are running 24 hours per day and have clocks inside them, why not give them the capability of kicking off certain jobs at a predetermined time? This is especially important in departmental computing environments where you often do not have operators working around the clock. Of course, these job submission utilities were designed to run large user batch jobs as opposed to DBA utilities (no one thinks of us). However, they work perfectly well for DBA tasks and save you from having either to work overtime or to dial in to run jobs in the middle of the night.

In the UNIX environment, you will find two key utilities that support automated job submission (see Figure 24.1). The first utility is `at`. I use this to submit jobs that need to run at a certain time, but not on a regular basis. For example, suppose that you want to run a special tuning report after a special batch of software testing completes, but before the nightly backups kick in. When you submit a job through `at`, it is run once at the specified time. There are a number of options to the `at` command, but the basic format is the following:

`at time command`

Figure 24.1.
UNIX automated job
submission routines.



For the time, there are a number of options. The simplest is to put in the clock time you want the job to run in military format (0700 for 7 a.m. or 1430 for 2:30 p.m., for example). You can use a.m. and p.m., but you have to get the syntax correct. For the command, you need to put in the fully qualified path and script name that are to be run. Remember, these need to be shell scripts or UNIX executable routines (SQL*Plus). You cannot run SQL scripts directly from the UNIX prompt. Here are some examples:

```
$ at 1430 sqlplus system/manager /users/jgreene/tunecheck.sql
```

```
$ at 0300 tunecheck
```

You can use `at` to submit your daily list of tasks; however, you have to schedule them each day. There are a lot of people who want to schedule jobs to run over and over, so UNIX has a better utility for running these jobs: `cron`. The `cron` utility reads a set of tables, one for each user, that is stored in a specified directory on the system. These tables specify the list of jobs and when they are to be run. When `cron` finds a job in one of the tables that needs to be run, it starts the job under the user ID of the person that requested the job.

The key difference between `cron` and `at` is that `cron` enables you to use wildcard characters (*) to specify that a job is to be run every day of the week at a certain time or every Tuesday at a certain time. You can interactively add entries to your `cron` table, but it is easier to make up an ASCII text file that contains all your jobs and then use a command to update your `cron` table from that text file. The following is an example of how to format a `cron` table for DBA tasks:

```
# *****
#
# File:      /info/oracle/admin/bin/oracle.cron
# Purpose:   Store the cron table for the oracle user ID
#
# *****
# min|hour|day|month|day|script
#   |   |of mo|   |of wk|
#   |   |----|   |----|-----
# *****
#
50    06    *    *    0    /usr/tdi/bin/logswitch
25    10    *    *    1    /usr/oracle/admin/bin/tunecheck.batch.tst.util
25    10    *    *    5    /usr/oracle/admin/bin/tunecheck.batch.tst.tune
25    20    15    *    *    /usr/oracle/admin/bin/tunecheck.batch.tst.sec
#
# *****
# END OF TABLE
# *****
```

Let's go over the parts of this file. First, a header specifies what it is. Next, a header tells you what the various columns mean. The actual `cron` entries—the only items in the file without the comment indicator, #—follow. Finally, an end-of-table, end-of-script comment indicates that you have reached the end.

The four entries in the `cron` table can be translated as follows. The first one runs a shell script that copies the Oracle log files to a backup file and then cleans them out. This script will be run every Sunday (day 0) at 6:50 a.m.. The next entry runs the database utilization report on Monday at 10:25 a.m.. The third entry runs a database tuning report on Friday at 10:25 a.m.. Finally, the last entry runs a database security report on the 15th of every month at 8:25 p.m. (2025 in military time). As you can see, with the number of columns that you have, you can provide pretty good control over when various tasks are run.

Now that you understand the basics of `cron`, it is time to set up your own `cron` table. The first thing to do is have your system administrator put you in the `cron` and `at` allow files. If you are not in these tables (or these tables are not set up to enable all users to have access to these utilities), you will get an error message when you try to use `at` or `cron`. Once the permissions are obtained, there is one command with two options to remember, `crontab`. The first option is the `-l` option, which lists the contents of your current `cron` table. The second option is the `crontab` command followed by the name of the ASCII text file in which you have entered your `cron` table entries. When you use this second option, it replaces the current contents of your `cron`

table with that of the text file. For example, suppose you wanted to update your cron table with the one shown, which you have called `oracle.cron`. The sequence of events may go something like this:

```
$ crontab -l

# *****
# File:      /info/oracle/admin/bin/oracle.cron
# Purpose:   Store the cron table for the oracle user ID
# *****
# min|hour|day|month|day|script
#   |    |of mo|    |of wk|
# ---|---|---|---|---|-----
# *****
50    06    *    *    0    /usr/tdi/bin/logswitch
25    10    *    *    1    /usr/oracle/admin/bin/tunecheck.batch.tst.util
# *****
# END OF TABLE
# *****

$ crontab oracle.cron

$ crontab -l

# *****
# File:      /info/oracle/admin/bin/oracle.cron
# Purpose:   Store the cron table for the oracle user ID
# *****
# min|hour|day|month|day|script
#   |    |of mo|    |of wk|
# ---|---|---|---|---|-----
# *****
50    06    *    *    0    /usr/tdi/bin/logswitch
25    10    *    *    1    /usr/oracle/admin/bin/tunecheck.batch.tst.util
25    10    *    *    5    /usr/oracle/admin/bin/tunecheck.batch.tst.tune
25    20    15    *    *    /usr/oracle/admin/bin/tunecheck.batch.tst.sec
# *****
# END OF TABLE
# *****
```

One final, important note about using the `cron` and `at` utilities. When you submit a job through `cron` or `at`, it does not execute in the same environment that your personal logon does. UNIX has a separate environment set up for these jobs. Common problems with this environment include the following:

- ♦ You do not get any of the wonderful environmental variables (such as `ORACLE_HOME` and `ORACLE_SID`) that you have set up in your `.login` or `.profile` files. Therefore, when you build these scripts, you need to set these variables up in your shell scripts.
- ♦ You may not execute under the same shell when submitting jobs through `at` or `cron`. UNIX has three common shells (Bourne, Korn, and C). If you really want to ensure that your scripts run under the shell you want, start your shell scripts with `#!/ksh` or `#!/csh` (for Korn or C shell, respectively). It's annoying, but you get used to it.

There are a lot of options and tricks regarding submitting jobs through `cron` and `at`. In case you haven't figured it out, I have also worked as a system administrator and programmer in the past, so I have learned more than most DBAs need to know about some of these things. This section tells you what you need to get productive without weighing you down with details that you may never use. Some might argue that you know enough now to be dangerous. However, that is what the next section is about. It presents a few tricks that I use to develop and test scripts.

DEVELOPING AND TESTING SCRIPTS

Some of you may be very familiar with script development, but others may have never touched it. Although there are entire books on this topic, this book has a lot of other material to cover. Therefore, let's concentrate on a few tips for script development. The material may not be all-inclusive, but it should be enough to get you started.

Because scripts tend to be very small applications with a limited set of functions, I usually start by drawing out what I want on a sheet of paper. I list the functions that I want done and then organize them as to how I want to do them. This may not be formal documentation, but I use it to organize my thoughts before I start coding. When done, I have a flow of the tasks that the scripts are supposed to perform and a division of labor between scripts if I will be writing more than one to satisfy the need.

I usually go right into my text editor and start writing the software. I use `vi` because it is available on the many systems I work with even though there are better editors out there. If there is a particular function that I have not used before or I am unsure of how to do something, I usually go into `SQL*Plus` and try out the commands manually. Remember that if you get a particularly long and obnoxious command working the way you want it, you can use `SQL*Plus`'s `save` command to save the file to text and then concatenate this command onto the end of your script files.

Once the scripts are developed, you can go to your test instance to try them out. You can get away with testing reports against a production instance, but you should be extremely careful about update scripts being tested against a production database. The scripts are so sensitive to little typographical errors that it could be risky. Run `SQL` scripts manually from the `SQL*Plus` prompt. This is the environment in which you get the most information display regarding any errors or processing status. After you run the `SQL` script manually, you can execute them from the `UNIX` prompt using any shell scripts that you have developed. Finally, if they are destined to be run under `cron` or `at`, test them under these environments. Remember, `cron` and `at` do not use the same environment that you have when you are logged in at the `UNIX` prompt.

One final tip: The UNIX operating system has a little command, known as `ps` (process status), that enables you to determine the status of your processes and all the processes on the system. When you select the correct options, you get a listing of the user, what time the job started, some other details, and finally the command that the user executed. If you used SQL*Plus to run your scripts and typed the user ID and password of one of your privileged accounts in the command line, that user ID and password will be available for anyone on the system to look at as long as your job is running.

This could be a major security hole. There are two solutions to this problem. The first is to run the scripts under SQL*DBA if you do not need to worry about the look of the output. SQL*DBA provides very limited formatting services. The other option is to use a command line similar to the following:

```
sqlplus userid @script < password-file
```

This command is similar to those discussed previously except that you do not include the password next to the user ID and you have added a redirect of standard input (<) at the end of the line. Thinking about how SQL*Plus works interactively, you have the option of putting part of all the user ID and password on the `sqlplus` command line. SQL*Plus prompts you for what you do not fill in. The format shown takes advantage of this function because you specify the user ID, so SQL*Plus prompts for a password. However, you have redirected the standard input from your keyboard to an ASCII text file that contains a single line, which is the password for the Oracle user ID that you specify on the command line. When you run a `ps` while this command is running, you see only the `sqlplus` command, the user ID, and the script name. One final note: you need to ensure that this password file is protected so that only the right people can read it (that is, 400 or 440 permission under UNIX). Other options include hard-coding the Oracle user ID and password at the beginning of the SQL script or using accounts with operating system authentication and putting only the slash (/) in the command line.

MONITORING THE RESULTS

So far this may have been a pretty challenging chapter, especially for those of you who have not dealt with UNIX for very long. There is one more topic to explore to help you develop sound scripts—checking to see what your scripts have been doing. There are three common options to consider:

- ♦ Writing the results to a log file
- ♦ Writing the results to a log file and then using electronic mail to send it to the right people
- ♦ Printing the results

The most basic technique is the log file. You have two options for log files. First, if you use SQL*Plus to run your scripts, you can spool all the output to an ASCII text file. Then, when the script is done, you can review this text file to ensure that the script did what you wanted. The second option, which is more useful if you run a series of SQL scripts and just want a single log file output from the job, is to redirect the output of your job to an ASCII text file. When you run a job from the command line, it sends the output to your terminal. A job submitted through `cron` or `at` does not have a terminal to which to go. Normally, this output is just lost. However, if you use the output redirection operator (`>`), you can send the output to a text file by using a command similar to the following:

```
sqlplus system/manager @/user/jgreene/tunecheck >/user/jgreene/tune.log
```

An extension of this technique that helps when you are busy is to create a log file and then send it to yourself via electronic mail. This saves you from having to remember to check when your schedule jobs are completed and remember where you put the log files. The command that used to accomplish the function is the `mailx` command that is formatted as follows:

```
cat log-file-name | mailx -s subject-line recipient
```

Basically, you use the UNIX `cat` command to list the file. You then pipe (`|`) the output into the `mailx` utility. The `-s` option enables you to specify a subject line to be displayed for the recipient's mail reader. Finally, you need to list the recipients' electronic mail addresses (which are usually their operating system logon IDs). I have used this type of script to send e-mail to remote UNIX servers (so that I had to log in to only one system to monitor all my systems). I have even seen it linked to LAN-based e-mail systems such as Microsoft mail (in which case the mail icon on your desktop flashes when you get mail). Here is an example of using the mail to send the output of a tuning report to a local UNIX mail account:

```
cat tune.log | mailx -s Tuning-Report-Done jgreene
```

A final option is to print the contents of the log file automatically. The simplest way to do this is to include in your shell script the line printer (`lp`) command similar to the following:

```
lp log-file-name
```

SUMMARY

There are a number of possibilities to make your scripts more powerful and friendly. These scripts can help take some of the burden off of you. They can save typing repetitive tasks and also serve to remember to run some of those monitoring reports that you might not find time to run on busy days. This chapter provided a quick

introduction into what is possible. The scripts on the enclosed disk demonstrate these techniques and some additional ones.

This chapter also concludes Part 4 of this book, Developing a Database Administration Scheme. By now, you should feel comfortable with some of the planning processes that define how you are going to run your databases. Much of this needs to be customized to your particular environment. With the list of things to consider and general guidelines in this section of the book, you should be able to plan your ground rules.

The next section of this book covers the daily routine—“how to” knowledge on the tasks that you will be performing regularly for your users. This material is important because how well you perform these tasks will determine how much time you have left over to tackle the big jobs that arise.

- 
- The “Typical” Day
 - User Account Maintenance
 - Tablespace Maintenance
 - Tabel and Index Maintenance

P A R T V

The Daily Routine

- [illegible]

This chapter begins Part 5 of this book, focusing on the day-to-day tasks that you will be asked to perform as an Oracle DBA. In addition to this overview chapter, there are chapters on the three most common routine maintenance tasks that you will confront. The first involves *user account maintenance*—adding new users, changing access permissions, and so forth. The next is *tablespace maintenance*. Because tablespaces map to data files that determine your available data storage space, these are the components that usually require the most intense space management. The final task involves *table and index maintenance* for good measure. Taken together, these should cover most of the routine tasks that come up during your day.

This particular chapter is devoted to an overview of the daily routine. Perhaps routine is a bit of a misnomer—there is usually a lot of variety in the requests and problems that are fielded during a given day. However, a brief overview is useful for those who have not yet had the pleasure of serving as a DBA. You also will be introduced to some of the different environments in which I have worked to give you a feeling for the possibilities that are out there.

Your daily schedule depends heavily on the type of database you are working on and where you stand in the project life cycle. However, there are a few basic tasks to get you going while you drink your coffee. These tasks are designed to bring you up to speed on what is happening and get you focused for the day. My typical startup includes the following:

- ◆ Log into the system and make sure that everything is working correctly. Run a simple command that I store as an alias in my UNIX `.profile` command that executes a command to show me the Oracle processes that are running. Look for the four common processes of Oracle 7 instances (DBWR, LGWR, PMON, and SMON) and for other processes that I may need, including the archiver and SQL*Net listeners. The command that I use under UNIX is something similar to the following:

```
ps -ef | grep oracle
```

- ◆ Next, I look at my e-mail. I like to have my automated reports sent to my e-mail account rather than printed. I scan each of these reports, and after a while, I know exactly where to look to see the problems that are likely to appear in a given instance (for example, running out of space in a data warehouse or number of extents about to be exceeded in a transaction processing environment).

Tip

You can reduce your paper burden and turnaround times by redirecting your automated reports to your electronic mail account instead of printing them.

The next thing that I may review are the log files. Typically, if the instance is running smoothly, I do not check these files often. However, if the instance has been having problems with its processes or other similar problems, I check the log file to verify that everything is still working correctly. Some users may not notice whether a portion of their jobs fail when they are run overnight. I routinely look for the existence of .trc files in the bdump directory. Otherwise, I review these files only when a problem arises.

By the time I finish this process (and often *before*), the telephone is ringing or someone is stopping by with a question or problem. Most of these questions and problems can be solved quickly. There are problems or questions, however, that can take up an entire day. This is where the art of prioritization comes in. You have to learn to put routine issues politely in a queue. Some users have a tendency to walk into your office as opposed to sending you an e-mail because they feel that they will get your attention and get what they want completed immediately. This affects your ability to get what is most important accomplished, so you have to try to solve real problems first and put the others in a queue.

In production database instances, the rest of the day tends to involve working off a series of service requests that have been submitted by the users. In large instances, you may wind up devoting a fair amount of time to adding users and changing their privileges. Many of your tasks, such as de-fragmenting a table or tablespace, have to be performed during off hours (see Chapters 27 and 35 for more information about de-fragmentation). Therefore, you may have to fill your day with less important tasks and then work overtime. In production instances, I have found that it pays to be very deliberate when performing your tasks. DBAs have the power to do a lot of damage if they rush too quickly and make mistakes. Think each command though and proof it twice before you execute it in a production instance.

A development instance presents a different series of challenges for the DBA during the day. The volume of questions about how to do things tends to be much higher. Developers also typically request additional support in the form of privilege grants and creation of new database objects (if the DBA creates test database objects in your environment). Finally, developers are more likely to find new problems in the Oracle software, so you can usually expect to have to spend more time on the phone working with Oracle technical support.

One of the more challenging environments to work in as an Oracle DBA is the engineering or business environment where you have to serve as a part-time DBA. This usually means that you get a call for support or an Oracle problem when you are in the middle of doing your own development tasks or other work. The challenge is to make the time to support your compatriots while getting your other work done. Part of this comes with an understanding of what it going on combined with some experience in Oracle. You have to know when it is reasonable to tell someone to look it up in a book and when you have a serious problem.

SCHEDULED EVENTS

One of the first things that I worry about after I have checked to see that the instances are up and running is my calendar of scheduled events. I keep a personal calendar on which I write the DBA tasks, meetings, and dentist appointments. If you do not keep a personal calendar, perhaps a wall calendar or some other format will help you keep track of these major events. Here are some items to keep on a long-term calendar:

- ♦ All development milestones, including start of design, data structures complete, software development begins, software testing (as many phases as are used at your location) begin and end, and, of course, delivery of the new software
- ♦ All major planned database activities, such as software upgrades
- ♦ Time scheduled for maintenance tasks, such as tablespace de-fragmentation
- ♦ Major events in the life of the users, such as year-end closing of the accounting books

It is also useful to keep a listing of the routing jobs that you have scheduled to run (a printout of the `cron` table). This is just a routine check to make sure that you are receiving your scheduled reports and that tasks such as log switches are being completed as scheduled. This little attention to detail provides system changes and problems from preventing important jobs from running. Remember, you may also be relying on those same scheduling utilities to run your nightly backup scripts, upon which everyone depends.

One of the challenges in scheduling your time is to look not just at what is going to happen today, but a little further ahead so that you have a good feel for what is pending. As the due dates get closer, you should ask yourself whether you are ready to support these new events. If you are not ready, you should set aside time and schedule tasks to ensure that you are ready. The DBA job is a support role; users and developers appreciate having at least one person in the meeting ready to support them.

MONITORING

You can think of monitoring as an executive in charge of the database. The database does the work and you just sit back and watch. However, like a good executive, you have to make sure that your subordinates are doing what they have been assigned to do. In Chapter 30, you explore a number of monitoring activities and reports that you can run to make sure everything is functioning as planned. However, at this point, a quick list of things that you will be routinely monitoring include the following:

- ◆ Tablespace free space
- ◆ Table and index fragmentation
- ◆ Tuning of the database
- ◆ Privileges granted to various individuals
- ◆ Overall configuration of the database
- ◆ Status of the Oracle processes (is the database operational?)

If you set up your reports to run via automated scripts, you will typically wait for the printed report or electronic mail to come across your desk. The art that you have to master is being able to read these reports quickly to determine whether everything is okay so that you can go on to your next task. It is a balancing act. Experience will tell you where the most likely problems will occur. You will get used to the normal values that you will be seeing so that you will quickly know when something is wrong.

There are third-party products that are getting fairly sophisticated at real-time monitoring of Oracle databases. Oracle's new Server Manager product even shows promise to enable you to monitor multiple instances from a single location easily. If you subscribe to any of the database or Oracle publications, your name will probably get on a mailing list. Soon, you will be inundated with literature from companies selling products to help monitor Oracle databases. Perhaps some day, a company will offer a product that meets all your needs at a price you can afford.

USER SUPPORT

One of the most variable aspects of your day is user support. You never know when someone else will come up with a question or how difficult that question will be to answer. The level of support that you can afford to give and are expected to give will be controlled by the type of instance and organization. You also have to decide whether you are going to get into the business of answering detailed technical questions for developers.

In most of the production instances that I have worked on, the DBA has to field any and all questions from the end users. The trick is to know when to defer an application-related question to the appropriate developer. You may know something about the application, but you are likely not the expert and there are probably other things demanding your attention. If it is database-related, you should be able to give some sort of answer to the users. One of the tricks is to gauge the level of understanding of the users to whom you are speaking so that you do not thoroughly confuse them with a detailed discussion of Oracle internals when all they wanted was a single yes or no answer.

In development instances, you may well be the best resource to answer many of the questions that come up during the development cycle. Certainly, questions regarding the configuration of the instance and tuning normally fall within the scope of the DBA. However, you may be asked about efficient algorithm and query design. If this is in your job jar, as seen by your management, or it is just a quick question, try to field it. If it is not in your job description and it will distract you from important tasks, try to defer the question to someone who is responsible for these areas.

Part-time DBAs can often find user support to be the greatest challenge. You are usually one of them, working side-by-side. You have both DBA and other duties that fill up your day. The trick is to help your compatriots find the answers without your having to do all the work for them. If you have technical support contracts, perhaps you can defer some of their questions to them.

An important technique that you have to learn is deciding when to ask people to wait until you can get back to them. You need to ensure that the answers to minor questions do not prevent you from completing higher-priority tasks. Prioritization of tasks can be an art. Asking someone to wait in a diplomatic fashion can also be a trick with certain people. However, because most DBAs have far too much to do, it is often an essential set of skills.

PROBLEMS

Suppose you have your days well planned. Your automated monitoring tools are feeding you the routine reports that you need to keep track of everything. You have a detailed long-term schedule that you are following like clockwork. Suddenly, the instances are crashing around your feet, developers are reporting errors that you have never heard of, and vice presidents want to know what's wrong. Maybe that's a little far-fetched, but you get the idea. The key issue here is dealing with problems.

Part VII of this book is devoted to dealing with problems. For now, you get a brief overview.

Any time you have a problem, it is important to gather all the data that you can about it. Write the data down, even if you just use a plain notebook. What seems like a simple problem may turn into a big problem when you get further into it. When you have big problems, you usually wind up calling Oracle Technical Support and it needs detailed information to track down exactly where the problem is. If you do not have this data available, you may wind up having to spend time re-creating the problem. Worse yet, Technical Support may not have enough data to give you a recommendation and you may have to wait until the next time the problem occurs.

Once you write down the basic errors and symptoms, you need to determine the *real* cause of the problem. There are a number of Oracle error messages that do not tell you the root cause of the problem. They often detect only *symptoms* of the true problem. A classic example from the data warehouse world would be when you get a "failed to allocate extent of size x in tablespace y" error message. What does this really mean? It could mean that the tablespace is out of space. It could mean that you exceeded the number of extents limitation on the particular table. Most deviously of all, it could mean that there is plenty of space in the tablespace, but you do not have a large enough contiguous extent to meet this need.

Now that you have tracked down the real cause of the problem, you are faced with a decision. If it is a minor problem, do you work on it now or do one of the other hot tasks that you promised to complete by the end of the day? Some people from pure production shops would probably say that any problem is the top priority, but what if you are a development shop and you find that people need more space to build a test table? Can you ask them to test with a smaller data set until tomorrow so that you can complete a task on the production instance? Prioritization can be tricky because it is quite possible that someone will be unhappy.

When you do work on the problem, you should have a plan of attack ready. If the problem is beyond your capabilities (a bug in the Oracle software, for example), you may have to call in outside support. If you believe that the problem requires outside support, you may want to call it in as quickly as possible. The support hierarchy of most vendors, including Oracle, usually asks you to prioritize the problem. I have gotten a call back within an hour on most down production instance calls. However, for problems that are just annoying, you may wait a day or so for a response. You need to factor that into your timeline and also make everyone else aware of it.

Most users and developers do not understand the support structure with which you have to work. They believe that you have infinite amounts of immediate support that tell you to type a few lines and then the world's problems are solved. They do not understand the difference between a table that needs to be de-fragmented and a major bug that needs a patch to get working. Your job, in addition to solving the problem, is to make sure that everyone understands what is going on and some expected timelines.

Finally, you may consider documenting in a log the problems that you encounter. I usually staple all my notes on a particular problem together and store the data in a file folder. This can come in useful later if you have the same problem on another instance. It can also help out friends when they run into the same problem. They will then owe you one and help you out in the future. A local network of Oracle DBAs can be a useful weapon to have in your arsenal.

To summarize, troubleshooting can be broken down into five phases:

- ◆ Capturing all the data related to a problem
- ◆ Determining the real problem
- ◆ Prioritizing the problem to see when you should work on it
- ◆ Solving the problem
- ◆ Documenting for future reference (even if it means merely storing the sheets from your notepad in a file folder)

IF THERE IS ANY TIME LEFT

It's time for some positive thoughts. The following is a list of the things that you may want to think about doing if you get your daily job jar emptied:

- ◆ If you have internet access, check the Oracle DBA Internet Newsgroup (`comp.databases.oracle`) to see whether anyone else has answers to some of the questions that have been bugging you. If nothing else, you may find some people who have more problems than you do.
- ◆ If you want to look at how vendors are positioning their products, you can look at the mailings that you are getting from companies that have purchased mailing lists from some of the magazines to which you subscribe.
- ◆ Speaking of magazines, you can take a stab at what is typically a mountain of back reading that keeps piling up on your desk while you have other work to do.
- ◆ You could look at your long-term schedule to see how well you are preparing for some of the major milestones that are looming.
- ◆ You could go through your Oracle data directories and look for some of the excess files that might be out there. A classic example is the export file that you created when you de-fragmented a table two weeks ago but forgot to delete.

As has been mentioned several times in this book, there are some folks who do not like to write things down or deal with paperwork and others who make lists and need to keep track of what is going on around them. Use the following checklist for daily tasks.

CHECKLIST

FIRST THING IN THE MORNING

- ☐ Check that all Oracle processes are running.
- ☐ Review e-mail and reports generated overnight.
- ☐ Review log files if problems suspected.
- ☐ Can I connect to the instance and run a simple query?

PLANNING FOR THE DAY

- ☐ What are the scheduled events for the day?
- ☐ What routine jobs will be run today?
- ☐ What events are coming up that I should be preparing for?
- ☐ Review monitoring reports.
- ☐ What tasks do I hope to get accomplished today (service requests and other tasks)?

AT THE END OF THE DAY

- ☐ Are the Oracle processes still running?
- ☐ Can I connect to the instance and run a simple query?
- ☐ Is there any coordination required for jobs scheduled to run tonight?
- ☐ Did I leave my PC or terminal logged in?

SUMMARY

This chapter was geared toward new DBAs who might benefit from a few words of experience and a feeling for what is facing them. For those of you looking for the technical details behind the daily tasks, don't worry—they are coming up next. In the next several chapters, you learn about the three most common routine tasks that I have been called upon to do as a DBA:

- ◆ User account maintenance
- ◆ Tablespace maintenance
- ◆ Table and index maintenance

- 
- User Maintenance and the Security Scheme
 - Using System Logon IDs for Access
 - Adding New Users to the Database
 - Changing User Access Rights
 - Deleting Users from the System
 - Temporarily Disabling Users

CHAPTER 26

User Account Maintenance

Many of the things that you will do as a DBA occur behind the scenes. Most of your users will not know what you are doing or why you are doing them. (Perhaps that is why we feel so unappreciated at times.) This chapter deals with the tasks that users *will* notice—the common tasks associated with giving the users access to your database. There are three levels of access:

- ♦ Ability to connect to the instance
- ♦ Ability to perform special functions, such as creating tables or views
- ♦ Ability to access data in your tables and views and the ability to access other database objects, such as stored procedures

This chapter discusses the common tasks that you will run across—adding users, modifying their privileges, and revoking their access. Oracle, of course, has a wide range of options available for most of these commands. This chapter does not cover all of the options. If you want to try something fancy, look at the SQL Language Reference to see all the command options that are available to you. This chapter sticks to the basics that you need in your daily job.

This chapter contains many examples of the commands that you might use. You can take these commands, fill in the details unique to your needs (such as user ID, role name, and so forth), and go from there.

Finally, in this chapter you explore some of the issues that you should be thinking about when you are performing user account maintenance. This subject ties in very closely with data security, so you need to ensure that you have a sound plan for controlling access and that you follow this plan. When you have options available to you, the pros and cons of each of the options are presented. Based on this discussion, you can choose the options that make the most sense for your particular situation.

USER MAINTENANCE AND THE SECURITY SCHEME

There are many different types of requests that you may encounter related to user account maintenance. Far too many of them need immediate attention. For purposes of this discussion, let's break these requests down into three major categories:

- ♦ Adding users
- ♦ Modifying user access rights
- ♦ Deleting users

Tip

Although user administration tasks are usually relatively simple and can often wait for several days, they can be the user's first impression of the system and the computer support staff. Consider placing these tasks relatively high in your task list—they will not really crimp your schedule and they can give the users a favorable impression of your group.

Once again, Oracle provides a number of different alternatives for you to choose from in your user management activities. There is a series of television commercials that talks about how nice it is to have alternatives. However, when it comes to database administration, you need to choose from the alternatives and implement a consistent scheme for maintaining your user accounts. It is just too hard to keep things going when you have to remember which way you did a particular user's account. Therefore, it is important to develop a security scheme in advance, as you learned in Chapter 13.

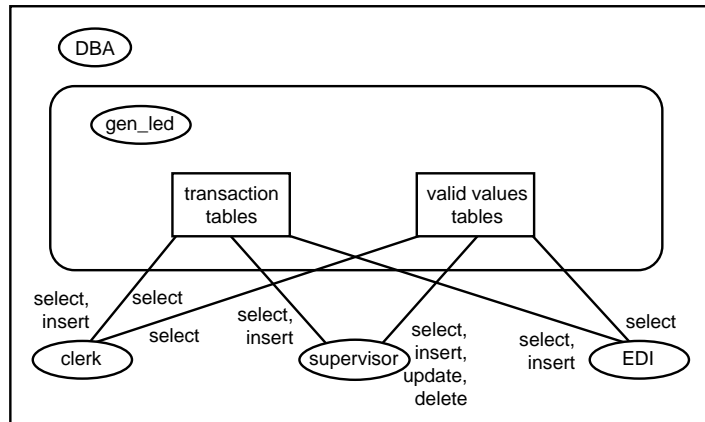
Let's quickly review the concepts of this security scheme. You start with your user and application requirements for access to data. You then build up a list of groups of people that will have the same privilege sets based on their job needs. If you are using roles in Oracle 7, you build roles associated with each of these groups of people. You then assign system and object access privileges to each of these groups based on their needs. Finally, you create user accounts and assign privileges, either directly or through the roles that you have set up.

To refresh your memory, let's review the example transaction processing system you learned in Chapter 13 (see Figure 26.1). This scheme groups everyone into five different roles. All privileges are assigned to the roles. Users acquire access privileges based on their being assigned to one or more of the roles that are established. This is a simple and highly maintainable scheme, especially when you expect to have a large number of users. Obviously, each application requires some thought as to how you implement its security scheme. Remember the following regarding security schemes:

- ◆ Use roles whenever possible. It greatly reduces the number of grants that you have to make and monitor on the system. I sometimes wonder how I was ever able to maintain a security system under Oracle 6 with all those individual privilege grants.
- ◆ Make sure that you break your groups down to the lowest level required. If you have a user who is not entitled to all the privileges of a group, you either have to make a new group or grant individual privileges to that user. If you have a number of fundamental groups, you can easily assign a user to multiple groups where that is appropriate.

- ◆ Keep the scheme as simple as possible. This may seem to compete with the last point, but it is a necessary tradeoff. Don't break things down to the point where you are creating groups that have individual object or system privileges.
- ◆ It is usually useful to draw out the privilege sets on a sheet of paper for your own reference later on. You can post this on your wall or keep it in a folder. You can even send it to the developers and key users so that they have a feeling for how security is being handled.

Figure 26.1.
Sample transaction
processing security
scheme.



To summarize this process, you should develop a security scheme before you begin administering your user accounts. This security scheme takes into account the application and user security needs. It provides a framework for you to create new user accounts quickly that have the appropriate privileges. It should have the benefits of making the maintenance and monitoring burdens of your job much easier. Once you have established this scheme, it is your challenge to remain consistent with it. You should not deviate from plans unless it is really necessary.

USING SYSTEM LOGON IDs FOR ACCESS

An important concept to consider when planning how you are going to implement user accounts is how the users will identify themselves to the system via passwords. By now, you may be getting sick of having to deal with all the options that are presented to you. For authentication, Oracle provides you with two basic options. You can set up Oracle user IDs to require an Oracle password or to be linked to the operating system user ID (therefore requiring no passwords).

The first option is what you might expect. You create an Oracle user ID and assign a password to it. Users are required to enter their user IDs and passwords before being permitted to access the Oracle system. Users are permitted to change the passwords, although there are no sophisticated utilities to validate the complexity of the passwords or to require the users to change those passwords routinely. My experience is that they rarely change their Oracle passwords and this can present a security problem.

The second option is to have users identified by their operating system user IDs. In effect, you generate an account that says if Joe is logged onto the operating system as `jgreene`, let him into Oracle under the `ops$jgreene` account without asking for another password. This was probably initially developed because users were complaining about the number of passwords that they were being asked to remember (the LAN operating system password, the UNIX password, the MVS password, the Oracle password, and so forth). However, it does present a number of opportunities for the DBA:

- ◆ The burden of maintaining passwords is placed on the operating system. You probably have enough to worry about.
- ◆ Operating systems tend to have built-in options to ensure that the passwords cannot be guessed easily and are changed routinely. Therefore, the operating system passwords are probably more secure than the Oracle passwords.
- ◆ Most users set the same password at the Oracle level that they use at the operating system level because they feel that it is too difficult to keep a large number of passwords straight. Therefore, if one password is compromised, they probably all are.
- ◆ It makes it very easy to build scripts that allow users to access your applications, because they can all be assumed to access Oracle directly.

To utilize the operating system authentication feature, you need to set up a parameter in your `init.ora` file that tells Oracle a prefix that you will use on all accounts that are to be authenticated from the operating system. Oracle needs some way to figure out which accounts will use this mechanism and this is the way they have chosen to do it. The default prefix is `ops$`. Therefore, if my operating system user ID is `jgreene`, my Oracle user ID would be `ops$jgreene` to use operating system authentication with the default prefix. You have the option of changing this prefix, but I have never come across a reason that would make me want to change it. Unless you have particularly strong feelings on the subject, you may want to keep the default so that as others come in to support you, they can quickly figure out what is going on.

What about situations wherein you are using SQL*Net as your primary access mechanism in a client-server environment? Typically, you just use Oracle password authentication in this situation. You do have the option of assigning a password to the `ops$` accounts, which can be used to gain access through SQL*Net. Normally, you set up an `ops$` account with a command similar to the following:

```
SQL> create user ops$jgreene identified externally;
```

However, you do have the option of putting a password in this location with a command similar to this (although there are rumblings that Oracle will remove this capability in the future):

```
SQL> create user ops$jgreene identified by mypassword;
```

The users who connect through SQL*Net have to use the `ops$` prefix when they are asked to type in their Oracle user ID. This option can come in handy when you have users who access Oracle both in a client-server and host-terminal fashion.

One final topic that is somewhat related to this topic is the use of group and Oracle default IDs to do your work. Just as you can create a dummy Oracle user ID to own a group of tables and other objects, you can create a dummy user ID that can serve as a group account. You can then have every user who belongs to that group access Oracle through the group account. Oracle placed no restrictions on the number of users who can be logged in using a given account. However, when you do this you lose accountability as to who performed a certain transaction. Worse still, if a particular user process becomes “hung” (stops responding to the user but does not get killed), you must go into SQL*DBA and figure out which of the many users using the same group account is the correct one to kill. Trust me, killing the wrong user can cause a lot of heartburn. I like to assign individual user IDs, but you have to do whatever makes sense given your individual situation.

An extension of the discussion in the last section involves using the `sys` and `system` user IDs, which are provided as defaults in Oracle. The one with the `sys` ID owns most of the data dictionary and other key tables that make up the core of the Oracle system. You become the `sys` user when you connect internally within SQL*DBA. The one with the `system` ID owns a number of useful views on the system. I tend to avoid using either of these accounts; I usually set up a DBA-privileged account under my own user ID. That way I can create my own schema objects that do not get mixed up with the Oracle system objects. If I am doing a fair amount of application development at the same time, I usually make a separate account for myself with only developer privileges. When you are working long hours feverishly trying to get your application working, you may issue some commands by mistake that could be problems if you are using a DBA-privileged account. (I guess what I am saying is that I do not trust even myself.)

Tip

Always change the passwords for SYS and SYSTEM on your system from the default values. You would be amazed at the number of production Oracle systems that I have run across that still use the default passwords for these DBA-privileged accounts.

ADDING NEW USERS TO THE DATABASE

It all starts when you grant users access to your Oracle database. Until then, you own the instance for yourself, but it does no useful work. In this section, you learn how to add users to the database and provide them with access to what they need to get their jobs done. Let me emphasize again that the trick is to ensure that what you implement is consistent with your security scheme and is therefore maintainable. Inconsistency may result in a short-term gain, but it can turn out to be a long-term pain.

The first thing that you need to do is set up an ID for the user. The basic command to accomplish this is the `create user` command, which has the following format:

```
create user USERID identified by PASSWORD;
```

Of course, you could not get away without having a few options that you need to consider when executing this command. The first question is whether to supply a password or used the `identified externally` option:

```
create user USERID identified externally;
```

These commands create a user ID using the default parameters established for your instance. There are three optional parameters that you may wish to set up, depending on your local needs:

- ◆ `default tablespace` specifies where objects such as tables will be placed if they are created by the user and the user does not specify a tablespace.
- ◆ `temporary tablespace` specifies where temporary segments, such as those used on sorts to disk, will be created for users. Users do not have the option of controlling where these objects are created. They will use whatever is specified in their user account profiles.
- ◆ `quotas` specify the maximum amount of space that a given user can take up within a given tablespace. When you allow users to create objects within the database, you run the risk of one user filling up your entire database at the expense of everyone else. `quotas` solves this problem.

The next example shows how you specify the default and temporary tablespaces for a new user:

```
create user jgreene
identified by mypassword
default tablespace users
temporary tablespace temp;
```

For your next example, you create an account for a developer (jsmith). Restrict each developer to a quota of 50M in the users' tablespace, which is the only tablespace in which they can create objects. This command is similar to the following:

```
create user jsmith
identified by longpassword
default tablespace users
temporary tablespace temp
quota 50M on users;
```

For your final example, consider the situation wherein you give each group of developers its own tablespace. You do not want to get in the position of deciding how much space each group member can use; therefore, you put that burden on the members. You give them a tablespace with a given amount of space and they have to work it out among themselves as to how much they use. You could give each of them a quota for the full size of the given tablespace, but you would have to change it if you increased the overall size of the tablespace. To simplify things, you can use the unlimited quota option:

```
create user jsmith
identified by longpassword
default tablespace developer1
temporary tablespace temp
quota unlimited on developer1;
```

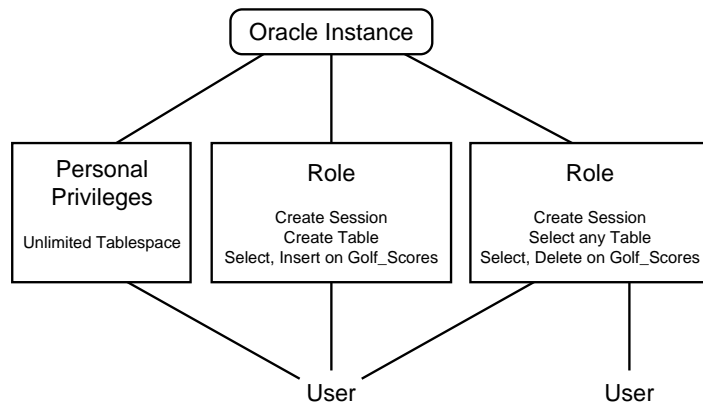
So far, however, you have created user IDs that are completely useless. They cannot even connect to the Oracle instance because you have not granted any system or object privileges to those users. Let me briefly review these concepts. In Oracle 6, you have three levels of access:

- ♦ *Connect* enables you to access but not create objects.
- ♦ *Resource* enables you to access and create objects.
- ♦ *DBA* enables you to do pretty much anything that you want.

In Oracle 7, you have a somewhat more complicated privilege scheme. For example, to get access to the Oracle instance, you have to give the user the `CREATE SESSION` privilege. To allow the user to create tables, you have to give the user the `CREATE TABLE` privilege. Because there are a large number of possible system privileges and this is complicated to administer, Oracle 7 gives you the option of creating roles to provide access rights to groups of similar users. You grant the appropriate privileges to the role and then grant the role to the users.

Finally, regardless of what version of Oracle that you are dealing with, you have to grant access to application data objects to users in order for them to be able to access the data. This is where roles in Oracle 7 is especially useful (see Figure 26.2). Consider, for example, an instance supporting Oracle Financials wherein you can have hundreds of tables that a user needs to access. The alternative is to develop a script that grants all the object access privileges to a user ID that you input interactively. However it is done, the requirement is the same. You have to understand your applications to know what accesses are required (select, insert, update, or delete) on the various database objects to get the different types of users functional.

Figure 26.2.
Roles and grants for users.



To create a role, you use a command similar to this:

```
create role developer;
```

To make this role useful, you need to grant privileges (either system or object) to that role:

```
grant create session to developer;
```

To grant access to a database object, you use a command similar to the following:

```
grant select,insert on this_table to developer;
```

After you complete this for all the roles in your security scheme (if you choose to use roles) and have created a user, it is time to grant some privileges or roles to that user. To grant individual system or object privileges to a user, you use a command similar to those shown that were used to grant privileges to roles. You merely substitute the Oracle user ID in place of the role name:

```
grant developer to ops$jgreene;
```


One of the problems that you will find, especially in instances that have somewhat complicated security schemes, is knowing which privileges to grant to a new user. What do you do when you get an e-mail saying to add jsmith to your Oracle instance? What privileges do you grant? I recommend that you formalize the process somewhat with a new user request form (either paper or electronic). This form does not have to be complicated, but make sure that the users understand it. If you ask them whether you should grant `ALTER ANY TABLE` to a user, they are likely to get confused. However, if you ask whether they should have the `ACCOUNTING_CLERK` role, they have a good chance of getting that one right. You also can put blanks on this form (or list the electronic signoffs) that are required to obtain approval for access by a specific user. You will find a sample user request form on the disk that comes with this book.

Very few users know how to change their Oracle password. Fewer still have any desire to do so. Therefore, when you set up a new user, you should carefully consider the password that you assign. If it is easily guessed (or the same for every user), the odds are that your instance will be easy to penetrate. If you make it too difficult, your users may have trouble connecting to the instance. Somewhere between lies a compromise that will work for you.

CHANGING USER ACCESS RIGHTS

As time rolls on, users will come and go. They transfer between departments and new applications are added to the system. Therefore, you will almost certainly have to change individual user account accesses. These changes fall into three categories: the need to add privileges to a user or role, the requirement to revoke privileges from a user or role, and the ability to change the roles to which a user is assigned.

The first of these requirements is simple enough. You need the ability to grant additional privileges to a role or user. This often results when new applications are produced or additional tables are created for existing applications. You may also get this requirement when a user or role needs additional system privileges. An example of this might be that your developers suddenly realize the benefits of views, so they all want the ability to create views. Oracle makes granting additional privileges to a role relatively simple. You use the same grant commands that you used to grant the initial privileges. The grant command is additive; you keep adding privileges to a user or role forever.

You may also need to revoke privileges or roles from a user. This could happen when a user is transferred to another assignment and therefore needs a different privilege scheme in the new job. To revoke a privilege from a user or role, you use a command similar to the following:

```
revoke CREATE TABLE from developer;
```


You use similar commands to add or revoke a role from a particular user. Here are some samples of these commands:

```
grant accountant to jsmith;
```

```
revoke developer from jsmith;
```

A few notes are in order here about changing user privileges in your instance:

- ◆ You can grant specific privileges to special users even when you are using roles. You should be careful about this in that it can complicate your security scheme a little bit. However, if you have only one developer who is allowed to create public synonyms, it is a waste to create a special role just for him. Instead, you can give him the standard developer privileges and then grant CREATE PUBLIC SYNONYM separately.
- ◆ There is no automatic privilege expiration in Oracle. If you give someone access to an object, that access remains in effect until you revoke it or drop the object.
- ◆ You should routinely check out the grants that are in effect for your instance and clean up those that are no longer needed. For example, you may know that a particular user left the company a month ago, but that account still exists on the system.

One final note regarding privilege changes. Sometimes it is helpful to check which privileges and roles a particular user has before you revoke or add privileges. These are the commands that you use to find the existing privilege sets for all users:

```
SQL> select * from dba_sys_privs;
```

GRANTEE	PRIVILEGE	ADM
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE DATABASE LINK	NO
CONNECT	CREATE SEQUENCE	NO
CONNECT	CREATE SESSION	NO
CONNECT	CREATE SYNONYM	NO
CONNECT	CREATE TABLE	NO
CONNECT	CREATE VIEW	NO
DBA	ALTER ANY CLUSTER	YES
DBA	ALTER ANY INDEX	YES
DBA	ALTER ANY PROCEDURE	YES
DBA	ALTER ANY ROLE	YES
DBA	ALTER ANY SEQUENCE	YES
DBA	ALTER ANY SNAPSHOT	YES
DBA	ALTER ANY TABLE	YES
DBA	ALTER ANY TRIGGER	YES
DBA	ALTER DATABASE	YES
DBA	ALTER PROFILE	YES
DBA	ALTER RESOURCE COST	YES
DBA	ALTER ROLLBACK SEGMENT	YES
DBA	ALTER SESSION	YES

GRANTEE	PRIVILEGE	ADM
DBA	ALTER SYSTEM	YES
DBA	ALTER TABLESPACE	YES
DBA	ALTER USER	YES
DBA	ANALYZE ANY	YES
DBA	AUDIT ANY	YES
DBA	AUDIT SYSTEM	YES
DBA	BACKUP ANY TABLE	YES
DBA	BECOME USER	YES
DBA	COMMENT ANY TABLE	YES
DBA	CREATE ANY CLUSTER	YES
DBA	CREATE ANY INDEX	YES
DBA	CREATE ANY PROCEDURE	YES
DBA	CREATE ANY SEQUENCE	YES
DBA	CREATE ANY SNAPSHOT	YES
DBA	CREATE ANY SYNONYM	YES
DBA	CREATE ANY TABLE	YES
DBA	CREATE ANY TRIGGER	YES
DBA	CREATE ANY VIEW	YES
DBA	CREATE CLUSTER	YES
DBA	CREATE DATABASE LINK	YES
DBA	CREATE PROCEDURE	YES

GRANTEE	PRIVILEGE	ADM
DBA	CREATE PROFILE	YES
DBA	CREATE PUBLIC DATABASE LINK	YES
DBA	CREATE PUBLIC SYNONYM	YES
DBA	CREATE ROLE	YES
DBA	CREATE ROLLBACK SEGMENT	YES
DBA	CREATE SEQUENCE	YES
DBA	CREATE SESSION	YES
DBA	CREATE SNAPSHOT	YES
DBA	CREATE SYNONYM	YES
DBA	CREATE TABLE	YES
DBA	CREATE TABLESPACE	YES
DBA	CREATE TRIGGER	YES
DBA	CREATE USER	YES
DBA	CREATE VIEW	YES
DBA	DELETE ANY TABLE	YES
DBA	DROP ANY CLUSTER	YES
DBA	DROP ANY INDEX	YES
DBA	DROP ANY PROCEDURE	YES
DBA	DROP ANY ROLE	YES
DBA	DROP ANY SEQUENCE	YES
DBA	DROP ANY SNAPSHOT	YES

GRANTEE	PRIVILEGE	ADM
DBA	DROP ANY SYNONYM	YES
DBA	DROP ANY TABLE	YES
DBA	DROP ANY TRIGGER	YES
DBA	DROP ANY VIEW	YES
DBA	DROP PROFILE	YES
DBA	DROP PUBLIC DATABASE LINK	YES
DBA	DROP PUBLIC SYNONYM	YES

DBA	DROP ROLLBACK SEGMENT	YES
DBA	DROP TABLESPACE	YES
DBA	DROP USER	YES
DBA	EXECUTE ANY PROCEDURE	YES
DBA	FORCE ANY TRANSACTION	YES
DBA	FORCE TRANSACTION	YES
DBA	GRANT ANY PRIVILEGE	YES
DBA	GRANT ANY ROLE	YES
DBA	INSERT ANY TABLE	YES
DBA	LOCK ANY TABLE	YES
DBA	MANAGE TABLESPACE	YES
DBA	RESTRICTED SESSION	YES
DBA	SELECT ANY SEQUENCE	YES
DBA	SELECT ANY TABLE	YES

GRANTEE	PRIVILEGE	ADM
---------	-----------	-----

DBA	UPDATE ANY TABLE	YES
EXP_FULL_DATABASE	BACKUP ANY TABLE	NO
EXP_FULL_DATABASE	SELECT ANY TABLE	NO
GOLFER	CREATE SESSION	NO
GOLF_PRO	CREATE SESSION	NO
IMP_FULL_DATABASE	ALTER ANY TABLE	NO
IMP_FULL_DATABASE	AUDIT ANY	NO
IMP_FULL_DATABASE	BECOME USER	NO
IMP_FULL_DATABASE	COMMENT ANY TABLE	NO
IMP_FULL_DATABASE	CREATE ANY CLUSTER	NO
IMP_FULL_DATABASE	CREATE ANY INDEX	NO
IMP_FULL_DATABASE	CREATE ANY PROCEDURE	NO
IMP_FULL_DATABASE	CREATE ANY SEQUENCE	NO
IMP_FULL_DATABASE	CREATE ANY SNAPSHOT	NO
IMP_FULL_DATABASE	CREATE ANY SYNONYM	NO
IMP_FULL_DATABASE	CREATE ANY TABLE	NO
IMP_FULL_DATABASE	CREATE ANY TRIGGER	NO
IMP_FULL_DATABASE	CREATE ANY VIEW	NO
IMP_FULL_DATABASE	CREATE DATABASE LINK	NO
IMP_FULL_DATABASE	CREATE PROFILE	NO
IMP_FULL_DATABASE	CREATE PUBLIC DATABASE LINK	NO

GRANTEE	PRIVILEGE	ADM
---------	-----------	-----

IMP_FULL_DATABASE	CREATE PUBLIC SYNONYM	NO
IMP_FULL_DATABASE	CREATE ROLE	NO
IMP_FULL_DATABASE	CREATE ROLLBACK SEGMENT	NO
IMP_FULL_DATABASE	CREATE TABLESPACE	NO
IMP_FULL_DATABASE	CREATE USER	NO
IMP_FULL_DATABASE	DROP ANY CLUSTER	NO
IMP_FULL_DATABASE	DROP ANY INDEX	NO
IMP_FULL_DATABASE	DROP ANY PROCEDURE	NO
IMP_FULL_DATABASE	DROP ANY ROLE	NO
IMP_FULL_DATABASE	DROP ANY SEQUENCE	NO
IMP_FULL_DATABASE	DROP ANY SNAPSHOT	NO
IMP_FULL_DATABASE	DROP ANY SYNONYM	NO
IMP_FULL_DATABASE	DROP ANY TABLE	NO
IMP_FULL_DATABASE	DROP ANY TRIGGER	NO
IMP_FULL_DATABASE	DROP ANY VIEW	NO
IMP_FULL_DATABASE	DROP PROFILE	NO
IMP_FULL_DATABASE	DROP PUBLIC DATABASE LINK	NO

IMP_FULL_DATABASE	DROP PUBLIC SYNONYM	NO
IMP_FULL_DATABASE	DROP ROLLBACK SEGMENT	NO
IMP_FULL_DATABASE	DROP TABLESPACE	NO
IMP_FULL_DATABASE	DROP USER	NO
GRANTEE	PRIVILEGE	ADM
IMP_FULL_DATABASE	EXECUTE ANY PROCEDURE	NO
IMP_FULL_DATABASE	INSERT ANY TABLE	NO
IMP_FULL_DATABASE	SELECT ANY TABLE	NO
JGREENE	CREATE VIEW	NO
JGREENE	UNLIMITED TABLESPACE	NO
JSMITH	CREATE ANY VIEW	NO
JSMITH	CREATE TABLE	NO
PUBLIC	CREATE SESSION	NO
RESOURCE	CREATE CLUSTER	NO
RESOURCE	CREATE PROCEDURE	NO
RESOURCE	CREATE SEQUENCE	NO
RESOURCE	CREATE TABLE	NO
RESOURCE	CREATE TRIGGER	NO
SCOTT	UNLIMITED TABLESPACE	YES
SYSTEM	UNLIMITED TABLESPACE	YES

141 rows selected.

SQL> select * from dba_role_privs;

GRANTEE	GRANTED_ROLE	ADM	DEF
DBA	EXP_FULL_DATABASE	NO	YES
DBA	IMP_FULL_DATABASE	NO	YES
JGREENE	DBA	NO	YES
JGREENE	GOLFER	YES	YES
JGREENE	GOLF_PRO	YES	YES
JGREENE	RESOURCE	NO	YES
SCOTT	CONNECT	NO	YES
SCOTT	RESOURCE	YES	YES
SYS	CONNECT	YES	YES
SYS	DBA	YES	YES
SYS	EXP_FULL_DATABASE	YES	YES
SYS	IMP_FULL_DATABASE	YES	YES
SYS	RESOURCE	YES	YES
SYSTEM	DBA	YES	YES

14 rows selected.

DELETING USERS FROM THE SYSTEM

When a user leaves the company, you want to clean up that user's account and revoke all access to your corporate data. To accomplish this, Oracle has a drop user command. Before you learn this command though, let's explore what you are doing when you issue this command. First, you remove the user ID and therefore the ability for the user to connect to this database. You also remove any grants that have been given to this user. Therefore, if you change your mind and re-create the user, you have to grant all needed privileges again.

An important consideration is that every object in the database needs to be owned by some Oracle user ID. Therefore, if you try to drop a user ID that owns objects within the Oracle database, Oracle gives you an error message indicating that the user has objects. You can transfer the objects to another user (bsmith in the next example) with a command similar to this:

```
create table bsmith.addresses as select * from jgreene.addresses;
```

Once you transfer or delete all the user's objects, you can issue the drop user command, which has a simple syntax:

```
drop user jgreene;
```

If you want to save yourself the trouble of deleting a large number of tables that are no longer needed from a user who is about to be deleted, you can use the cascade option of the drop user command:

```
drop user jgreene cascade;
```

A few final notes about dropping a user. First, what you are doing is permanent. There is no rollback for these changes. Second, it is very useful to implement a process wherein you are notified when an individual leaves. Perhaps you can tie it in with the personnel department's checkout process. All they need to do is send you an e-mail when a person leaves. It will make your job easier and provide security against potential disgruntled ex-employees from accessing and damaging your systems.

TEMPORARILY DISABLING USERS

Suppose that you want to disable a user's ability to connect to the system, but do not want to drop the user's account. This situation may come up when a person goes on extended leave and your security administrator wants all access revoked until that person comes back. Perhaps the person is being fired and you want to prevent further access to the system, but you need time to go through that person's tables to see which ones are still needed.

The easiest trick is to revoke the user's ability to create a session. If you are granting explicit privileges to the user, all you have to do is revoke the CREATE SESSION privilege. None of the other privileges have any meaning if the user cannot connect to the instance. If your users inherit their capability of creating sessions from roles, you merely revoke all the roles that allow them to create sessions. A final option for user IDs that do not use operating system authentication is to change the users password to gibberish—you don't even have to write it down. You can still access the user's tables, thanks to your SELECT ANY TABLE privilege as a DBA.

SUMMARY

This chapter presented the first of the common daily tasks of the DBA. It discussed the services that you provide to allow users to access the Oracle database and specific data within the database, covering the options of the three basic services that you will provide—adding users, changing user privileges, and removing users from the system. Along the way, you learned the importance of keeping your user account maintenance procedures consistent with the security scheme that you have devised and picked up a few suggestions about user ID request forms and ways to disable users temporarily along the way.



- Care and Feeding of Tablespaces
- Monitoring and Planning
- Typical Problems and Their Solutions

CHAPTER 27



Tablespace Maintenance

Because data files map to tablespaces, you are actually managing your disk space when you are managing tablespaces. Given the rate at which business information is growing, you may find yourself having to closely monitor and routinely expand the size of your data files. This chapter presents the routine maintenance tasks associated with tablespaces.

Things rarely go exactly according to plan. It's a good idea to ensure that the unexpected changes stay within the safety margins (extra space designed just for this purpose) that you have worked into the plans. This is an area that can make many DBAs nervous. You realize that you need to ask for disk space because a new project took up more space than expected or business has just been unusually good. You want to avoid the lecture about how capital expenses need to be controlled and so forth.

This chapter deals with the “how to” part of the expansion of data files. It also discusses how to determine whether you need to add more space to a tablespace. Finally, it covers other routine operations that help keep your disk space cleaned up and ready for business. By the end of this chapter, you should feel comfortable with what it takes to expand or de-fragment a given tablespace (see sections later in this chapter and also Chapter 35 for more information about defragmentation).

CARE AND FEEDING OF TABLESPACES

The words “care and feeding” are used in the title of this section because supporting an Oracle database is analogous to keeping a pet around the house. You have to keep an eye on it, routinely look after its needs, and occasionally clean up after it. The keys to properly caring for an Oracle instance include the following:

- ♦ Planning out the space that will be required for the instance as it grows
- ♦ Routine monitoring of space utilization (both total free space and available extents)
- ♦ Expansion of the tablespaces by adding data files when needed (feeding)
- ♦ Defragmentation of the tablespaces to minimize the number of times that you have to add data files
- ♦ Reorganization of the tablespaces when needed to increase performance or better utilize disk space

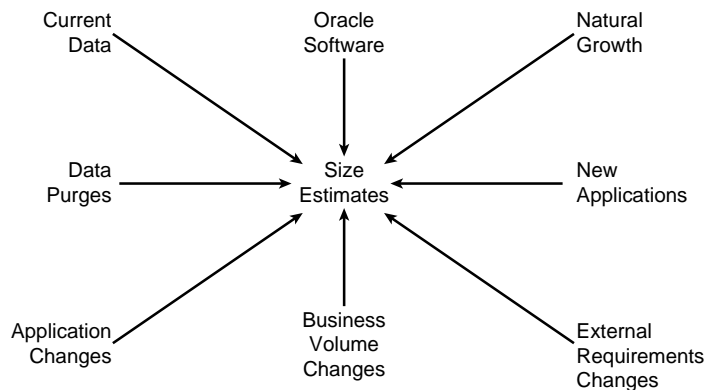
The first thing that you need to do in order to be proactive with your space management is to develop a plan showing how you expect disk space usage to vary over time. For some lucky folks, the answer may be that the size of the data will not vary because you will purge old data to fit within your disk limitations. Most of the rest of us face the fact that online data storage has been growing at a rather incredible rate for several decades now. There is no reason to assume that this pattern will change.

In most organizations, the DBA is asked to prepare a disk space projection in time for the project or data center manager to submit capital appropriation requests during the annual budgeting cycle. This is the time when everyone is thinking about money, especially capital expenditures (nonlabor or raw materials). Computers make up the vast majority of capital expenditures in most nonmanufacturing organizations and therefore usually garner a special degree of scrutiny. Lucky us.

When you consider what it takes to make an accurate projection of disk space, you'll realize what a challenge it can become. The following is only a partial list of the factors that you need to take into account to achieve any degree of accuracy (see Figure 27.1):

- ◆ The amount of data currently stored on the system.
- ◆ The space required for the Oracle software, including having multiple versions to support different applications or the upgrade process
- ◆ Natural growth in the amount of data stored
- ◆ Effect of any planned purges of data (for example, removing data that is more than five years old)
- ◆ Space required for new applications, some of which may not have been accurately designed yet
- ◆ Space required for changes in existing applications
- ◆ Space required to support changes in business volume (if you are tracking orders and your business is booming, you may need extra storage space)
- ◆ Changes that are completely out of your control (for example, if a new law changes your record retention requirements from three years to five years)

Figure 27.1.
Factors affecting space requirement plans.



This list could go on at some length. There may also be a number of factors that are unique to your particular business needs and your organization. However, by now you should have the general idea. When making up a plan, your goal is to start with

what you are using now and try to list all the factors that may change your disk needs and when you expect them to take effect. Next to each of these requirements, you list an estimate of how much they are going to increase or decrease your disk storage needs.

That may sound easy, but in practice it may be quite difficult. Again, many organizations look carefully at expenses for new computer equipment, so they may well challenge the numbers that are used to justify new disk drives. Many of the assumptions that you start with are rough. Developers cannot provide details on the design before the project begins. Start with a set of assumptions about each of the factors that may affect your disk calculations. Samples of these might include the following:

- ♦ Orders will increase by 25 percent in accordance with the business plan prepared by the CEO.
- ♦ We will increase the retention of data from two years to five years per service request number 12345 from the marketing department.
- ♦ The new inventory tracking system will process 200 movements per day with a total of 52,000 items in stock at any given time.

Don't qualify anything with the words "I think that." Instead, use such items as business plans or service requests from the user departments as the basis of the justification. If you do not have any such data, you should at least start with an estimate of the number of business items that will be processed (items in stock and inventory movements), because most users are not tuned in to rows in tables. The key is that if someone is looking to take shots at your plan, spread out the reasons so that this person will have to challenge the plans of everyone in the company.

Once you lay out your assumptions for the plan, you need to build on them to come up with a number for the amount of space needed. The simplest approach to apply to a new system that has not been designed is to draw an analogy to an existing system. Use this technique only when there is almost no design information available. Your logic for this is to state something such as, "the Midwest sales forecasting system should be about the same size as that of the West Coast system." It is not precise, but if you cannot wait for the design to be completed to budget for the disk space, it may be all you have.

A slightly more precise form of calculation may be to use rough percentages in your calculations. You can draw these from sales plans or other business unit developed forecasts. The logic for this is to apply these percentages to the existing disk space requirements and come up with a new number. An example might be that if your retention requirement goes from two years to five years, you multiply your existing tablespace needs by 250 percent.

Finally, a more precise method is to apply expected increases to actual numbers of rows and space utilization per row. This takes into account two inaccuracies in the previous methods. First, because you allocate space to tables and other objects in whole extents that consume the space even when they are not yet filled, you have only an approximation as to how much space is being consumed. Second, some parts of the database may not increase as you increase the number of business items stored. For example, your list of legal payment types will stay the same regardless of the number of orders you receive. Therefore, you need to account for which tables will increase row counts and which will not.

This can be a time-consuming and detailed process, but if you need the accuracy, here is how to approach it. First, you get a count of the number of rows in all your application's data tables. Next, you get an estimate of the average amount of space consumed by a row in each of the tables. Both of these data items are obtained by issuing queries against the database as follows:

```
select count(*) from table_name;

select avg(nvl(vsize(column_1),0))
+avg(nvl(vsize(column_2),0))+...+avg(nvl(vsize(column_n),0))
from table_name;
```

As you might guess, this can be extremely tedious when you consider that there may be a large number of columns in many different tables. You have to perform these queries for each of these tables to get the data you need. A classic example of when you may need to perform this type of calculation is sizing a large data warehouse based on size data determined from a relatively small test instance. Here, even a ten percent error could cause an error of several gigabytes.

Once you have this row size and count data, the only remaining data item that you need is an estimate of the number of rows that you will have in production. This is usually relatively easy to tie to some business forecast (for example, number of orders expected next year, production forecasts, and so forth). It is important to remember that your goal is to do the best that you can with the information provided to you. If you have documented your assumptions and everyone has bought into them, you can at least bring that up when they wonder why things did not go as expected.

Let's take a relatively simple example. You have an application with two tables—orders and valid customers. You have built a test instance that contains a small number of test orders and all your customers in the valid customers table. You collect the data that you need with some queries similar to the following:

```
SQL> select count(*) from customers;
```

```

COUNT(*)
-----
      47

SQL> select count(*) from orders;

COUNT(*)
-----
      61

SQL> select avg(nvl(vsize(customer),0))+avg(nvl(vsize(sales_rep),0)) "Avg Size"
2  from customers;

Avg Size
-----
      18

SQL> select avg(nvl(vsize(customer),0))+avg(nvl(vsize(date_ordered),0))+
2  avg(nvl(vsize(amount),0))+avg(nvl(vsize(date_shipped),0)) "Avg Size"
3  from orders;

Avg Size
-----
      22

```

Of course, you format the output of the `Avg Size` column to include as many decimal places as desired, but these numbers should be good enough. The `ANALYZE TABLE` command is an alternative for users with Oracle 7.1 and later. This command calculates the average row size and number of rows for you. It even has the side benefit of allowing you to estimate sizes based on looking at a certain percentage of the total rows. This can be significant if you have a very large table to analyze. To review the results, merely look in the `DBA_TABLES` view. It could not be simpler.

Next, you input the numbers into a spreadsheet. You will save yourself a lot of time and headaches if you become friendly (or at least tolerant) of this tool. I recommend using spreadsheets because when management people see the amount of disk space required, they start changing the numbers around to achieve a more reasonable figure. You may find that they change the data retention requirement from eight years to five, for example, because they feel they will never justify a large amount of new disk drives. In either event, when these changes occur, you can quickly plug the changed numbers into your spreadsheet and print out the latest figures for them. The following is a very simple example spreadsheet made up with the numbers shown:

Purpose	Test Rows	Test Size\ Row	Est. Prod. Rows	Est. Prod. Mega- bytes	Number of 100M disks
Orders- tables	61	22	35000000	770	8
Customer- tables	47	18	47	0.000846	1

Customer- index	47	5	47	0.000235	1
Total				770.001081	10

Let's look at this spreadsheet closely. First, notice that calculations for the indexes and tablespaces are separated. This is because you will probably want to place them on separate physical disks and when you round to the next whole disk, you need to do it separately for the tables and the indexes. Also notice that the valid customer list size is not increased. This is because we assume that you already have all these customers in the existing test table. This is an extremely simple example, but it enables you to see the points without the confusion of two-page spreadsheets.

Factoring a margin of error into your plans is very important, but can also be politically sensitive. Some places say that they need to calculate everything exactly and you will just have to come back to them if you need more. This is an extremely risky position. It is impossible to calculate some of your factors exactly (how many things will be sold, for example). You also need to factor in that even if your purchase is approved in just a few days, it may take weeks or months to get the additional disk capacity installed and ready to create new data files. Therefore, build a column into the system that shows the amount of bytes, with a safety margin applied to the actual calculated disk quota (30 percent, for example). Another technique is to calculate the total number of disks that you need and then apply the safety margin as "hot spares." The following are examples of these techniques:

Purpose	Current Rows	Row Size	Est. Rows	Est. Size(MB)	With 30% Margin (MB)
Orders tables	61	22	35,000,000	770	1001
Customer- tables	47	18	47,000	0.8	1.04
Customer- index	47	5	47,000	0.2	0.26
Totals					1003

-- ALTERNATIVELY --

Purpose	Current Rows	Row Size	Est. Rows	Est. Size(MB)
Orders tables	61	22	35,000,000	770
Customer- tables	47	18	47,000	0.8
Customer- index	47	5	47,000	0.2
Total data				771
Hot spares (30%)				232
Total disk storage				1003

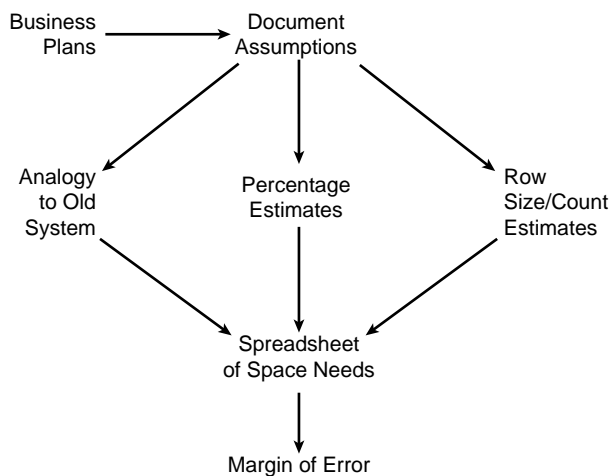
Here is a suggestion that might save you some time. In most of the applications that I deal with, I find a relatively small number of tables that contain most of the facts about the data that I am tracking (for example, orders in the previous example). These are the tables that grow over time and take up the vast majority of the space in an instance. If you have to use the more precise sizing techniques shown, perhaps you can get away with focusing on getting accurate sizes for the big tables and then

use a less accurate technique for the little tables that are not going to make much of a difference anyway. For example:

Purpose	Current Rows	Row Size	Est. Rows	Est. Size (MB)
sales table	1000	10	32,000,000	320
orders table	230	32	1,000,000	32
validation tables				3
admin tables				1
Total data				356

The overall process for developing a disk space plan is shown in Figure 27.2. Even if you do not have to do it for your budget cycle, it may be a useful exercise. If you implement this and track it against actual growth, you will be in a better position to recommend when additional disks are needed, before they become a crisis. Finally, it is important to update your plans routinely to reflect the inevitable changes that take place as projects are added or canceled.

Figure 27.2.
Overall space planning process.



MONITORING AND PLANNING

As alluded to late in the last section, monitoring and planning are intertwined. When planning becomes an ongoing process, you need to have monitoring set up to validate the accuracy of your plans. Oracle internally tracks hundreds of different parameters. The challenge is to figure out which of the many possible queries will quickly obtain the results that you want. This section describes my favorite method for accomplishing this.

The first thing that I like to do is create some views that make the SQL syntax much simpler. The first view that I create is `tablespace_alloc`, which shows the total size in bytes of each of the tablespaces. The other view, `tablespace_used`, shows the

number of bytes used by the various tables, indexes, and so forth. Here is the SQL to create these views, which I prefer to run as sys:

```
SQL> create or replace view tablespace_alloc
  2  as select tablespace_name,sum(bytes) Bytes
  3  from dba_data_files
  4  group by tablespace_name;
```

View created.

```
SQL> create or replace view tablespace_used
  2  as select tablespace_name,sum(bytes) Bytes
  3  from dba_extents
  4  group by tablespace_name;
```

View created.

```
SQL> create public synonym tablespace_alloc for tablespace_alloc;
```

Synonym created.

```
SQL> create public synonym tablespace_used for tablespace_used;
```

Synonym created.

This SQL can be found on the enclosed disk as file cr_dbavw.sql. Notice that I create public synonyms for each of these new views. Because I normally work in my personal account and do not want to have to type sys to get at the data, I create these views. In addition to using them for canned reports, I also use these views from the command line when working on problems. Once you have these views created, you can run an SQL query similar to the following to get a feel for how full your tablespaces are:

```
SQL> select u.tablespace_name,u.bytes,a.bytes
  2  from tablespace_used u,tablespace_alloc a
  3  where u.tablespace_name = a.tablespace_name;
```

Table_Space	BYTES	BYTES
ROLLBACK_DATA	409,600	3,145,728
SYSTEM	7,286,784	10,485,760
USER_DATA	92,160	3,145,728

Before I ran this query in SQL*Plus, I defined a format for the bytes column to include commas. This is especially useful in large databases wherein you need to know whether you have a gigabyte left or only 100M. I never can keep track of such things without the commas. If you have trouble keeping track of which column is which, you can add synonyms to the column headings to make it clear for you:

```
SQL> column used format 999,999,999
SQL> column alloc format 999,999,999
SQL> select u.tablespace_name,u.bytes used,a.bytes alloc
  2  from tablespace_used u,tablespace_alloc a
  3  where u.tablespace_name = a.tablespace_name;
```

Table_Space	USED	ALLOC
-----	-----	-----
ROLLBACK_DATA	409,600	3,145,728
SYSTEM	7,286,784	10,485,760
USER_DATA	92,160	3,145,728

Chapter 30 discusses routine monitoring in a little more detail. The views and queries presented here actually serve as part of a more inclusive report that shows a complete picture of space utilization.

Fragmentation is a condition wherein you have a number of disk extents that are not large enough to suit your needs. It occurs when you continuously add and delete extents of different sizes within your instance. After a while, you may find that you have a number of small extents free that add up to a significant amount of disk space. However, when Oracle allocates a new extent, it insists that the new extent fit in blocks of the disk that are located next to one another.

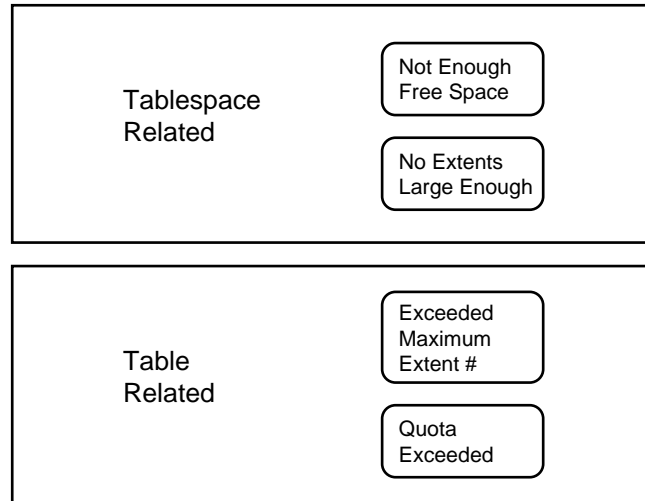
These are the two most common space monitoring issues that you will come across. Again, you should routinely bounce this against your disk usage plans to determine whether something is afoot and users are consuming more disk space than planned. It could be something as simple as having a developer make a huge test table and forget to delete it. However, it could be a signal that the user data volume is increasing unexpectedly and it is time to get users to reduce their data needs (by archiving old data) or get some additional disks ordered. Again, the goal is to manage your disk space proactively and avoid crises whenever possible.

TYPICAL PROBLEMS AND THEIR SOLUTIONS

If you ever look at the messages and codes manual for Oracle, you will realize that there are a large number of problems that could creep up on you. Most of them have a message that starts with `Failed to allocate extent...` Whatever the exact details of the message, there are four common things that cause you to fail when you try to add a new table or additional extents on existing tables (see Figure 27.3):

- ◆ There is not enough free space in the tablespace to create the extent that you want to create.
- ◆ There is enough free space, but there is not a contiguous extent large enough to hold the extent that you want to create (fragmentation).
- ◆ You have exceeded the maximum number of extents allowed (MAXEXTENTS) for the table or index.
- ◆ If you are creating a new table, you lack permission or have exceeded your quota in the designated tablespace.

*Figure 27.3.
Common reasons
extend allocation may
fail.*



For purposes of this discussion, let's consider the last two reasons to be problems with the individual tables. These problems are discussed in the next chapter. In this chapter, you address the first two problems, which deal with a lack of sufficient space in the tablespace for your needs. The first step is to determine whether you have insufficient space or fragmentation problems. The queries shown in the last section show you the total amount of space used versus the total amount of space allocated. From this, you can determine the amount of space that is free. You can even put this subtraction in your SQL `select` statement to have Oracle perform the calculation for you:

```
SQL> select u.tablespace_name,u.bytes,a.bytes,a.bytes-u.bytes free
       2 from tablespace_used u,tablespace_alloc a
       3 where u.tablespace_name = a.tablespace_name;
```

If you find that do not have enough available disk space, you need to add additional data files to the tablespace. The solution to this problem is simple if you have enough free disk space. You issue the `alter tablespace` command with the `add datafile` option. You need to give Oracle the fully qualified name of the datafile that you wish to create and the size that you wish it to be. Here is a sample of this command:

```
alter tablespace users
add datafile '/user1/oracle/prod/users02.dbf' size 100M;
```

If you find that you have sufficient space to create the extent that generated the error message for you, it is time to look at fragmentation. One note first, the error messages that indicate that Oracle failed to allocate an extent give the size of the extent in blocks. You have to multiply this figure by whatever your block size is (4096 bytes, for example) to determine the size in bytes. When you work with files, it is usually easier to express sizes in bytes, which the operating system understands.

The following is the query that will tell you the sizes of all of your free extents:

```
SQL> select free.tablespace_name,free.bytes
2    from sys.dba_free_space free
3    order by free.tablespace_name,free.bytes
4    ;
```

TABSPACE_NAME	BYTES
ROLLBACK_DATA	2734080
SYSTEM	3196928
TEMPORARY_DATA	2095104
USER_DATA	4096
USER_DATA	10240
USER_DATA	3051520

6 rows selected.

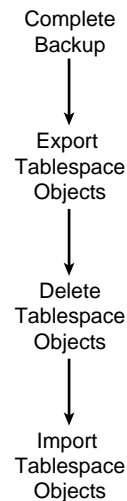
As you can see, there are three free extents in the USER_DATA tablespace. The last one is large enough to accommodate most needs (I have not created much in this instance other than examples for this book). However, the first two extents (which were example tables from this book that I deleted) are relatively small and may not be useful for most other tables that you would create. If you find that you do not have any extents that are large enough to store your data, you have to do something about it.

A logical question is how to get rid of this fragmentation. There are a number of third-party utilities out there that will move chunks of data around to achieve de-fragmentation of your tablespace. I have never used any of them. I usually resort to the utilities provided with every Oracle database[md]Import and Export. You perform a complete backup of your instance. You will delete things and, if you forget to export something, you will have nothing to import the data back into your instance. Trust me, always perform your backups before this major surgery.

Now that your backup is complete, it is time to start compressing extents in the tablespace, called *de-fragmentation*. The first step is to export all the tables in the tablespace. This makes a copy of your data in another location (disk or tape file). This export should be complete; you should normally select the `compress extents` option and export all related grants and so forth. The second step in this process is to delete all the objects that you have just backed up from the tablespace. When the tablespace is completely empty, you have a single large free extent. The final step is to run Oracle's Import utility to get the data back. This creates a series of extents that are contiguous to one another and you will be left with a single free extent at the end of your data file.

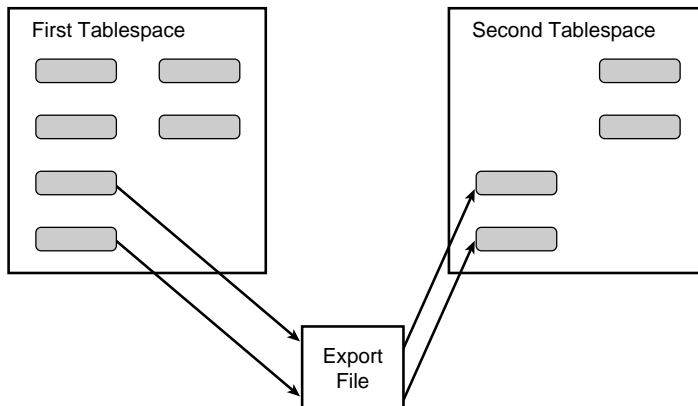
A few final notes on this process, which is illustrated in Figure 27.4. First, the largest single extent that you can have is the size of your data file. You cannot have extents that cross data files. Part of your fragmentation problem may be due to having a large number of small files. The solution to this is to reduce the number of data files so that you have fewer files that are larger. I have run across several instances where 2G files (the largest supported by the operating system) were created that consumed an entire disk drive. In this case, there is nothing more that you can do about the size of the data files. You should, however, be careful to choose extent sizes for your objects so that you can use as much of each individual file as possible. For example, if you have 2G files, choosing extent sizes of 1.1G wastes a large amount of disk space (after you create the first 1.1G extent in a data file, the remaining 0.9G is wasted because it cannot fit another one of your 1.1G extents).

Figure 27.4.
Tablespace
defragmentation
procedure.



There is an extension of the process described in the last section: tablespace reorganization. Once you have exported the objects from a tablespace, you do not necessarily have to put them back in the same place as they were stored before the export. You have the option of creating the objects (empty) in another tablespace and then selecting the option that Oracle Import will present you to ignore creation errors. It will not object to the fact that the object you are trying to import already exists. Instead, it will start filling up the existing object with the data from the export. Kind of a neat trick when you want to try to balance input/output load or free up space in a tablespace that is almost full although you have some relatively empty tablespaces. This process is shown in Figure 27.5, and is discussed in more detail in Chapter 28.

Figure 27.5.
Tablespace reorganiza-
tion process.



SUMMARY

This chapter presented some of the typical problems that you may encounter related to space in an Oracle instance. It started with a space planning process to help you proactively manage the disk space assigned to you. Then you learned the process for determining the nature of problems that might arise and the various procedures to correct these problems. As mentioned, there are a number of other problems that are possible with tablespaces. If you determine that the problem does not lie with a lack of space or fragmentation, please proceed to the general problem solving chapters starting with Chapter 34. However, the vast majority of tablespace problems that you will encounter during production operations deal with running out of space or number of extents. Between the discussion in this chapter and the next, you should be able to deal with these problems.



- Care and Feeding of Tables and Indexes
- Monitoring Tables and Indexes
- Typical Problems and Their Solutions
- Fragmentation

CHAPTER 28



Table and Index Maintenance

This chapter is the last of the chapters devoted to the daily routine. Because the main purpose of Oracle is to store data and the tables within Oracle are the basic storage units, this chapter on table and index maintenance is significant. The efficiency of your table design and how tables are arranged within your tablespaces have a strong impact on the performance of your applications. This chapter is designed to present some of the more common operations that you may be involved with as a DBA.

This topic is presented in a pattern that matches the life cycle of a table or index, starting with a discussion on the creation and planning aspects for the database objects. Next, you learn routine monitoring steps that enable you to see when the objects may be headed toward problems and the modifications that you, as the DBA, may be asked to make to these objects. Finally, the chapter presents some additional topics, such as reorganization of the tables, that may be needed to improve performance.

CARE AND FEEDING OF TABLES AND INDEXES

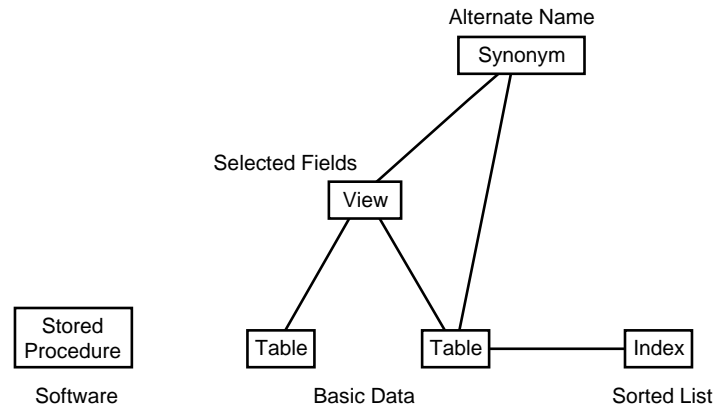
The performance of your applications depends on how well you design and maintain the tables and indexes in your system. There are, of course, a number of other database objects that you will work with—views, stored procedures, synonyms, and so forth. However, these items tend to be static over time because they do not store any actual data. The items that grow over time, and serve as the basis for these other objects, are the tables and the indexes that are associated with them. This section presents some of the planning and creation ideas related to these objects.

First, let's review how database objects fit together (see Figure 28.1). The basic unit of data storage is the table. To speed access to data in tables, lists that contain the columns that are commonly searched upon are created; these are known as indexes. To join tables together or select only a subset of the table data, developers can create views. To provide alternate names for the tables, Oracle supports the use of synonyms. Finally, Oracle enables you to store software within the database in the form of stored procedures. These are the basic database objects that are placed under the control of developers. Other database objects (such as rollback segments) are under the control of the DBA and will be addressed in later chapters.

Note

It is important to have a sound design. If your basic design is not sound, you will have great difficulty getting good, reliable performance from your instance.

Figure 28.1.
Overview of basic
database objects.



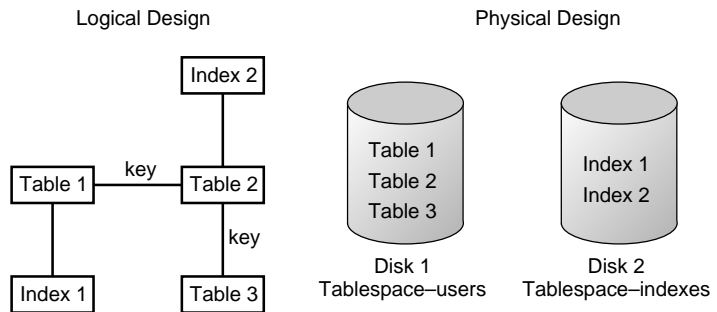
The process of building and maintaining sound tables begins in the design process. I have been in a number of different Oracle shops and have found that they vary widely in the level of participation that the DBA has in the table design process. In some shops, the DBAs are responsible for the entire table design (and even applications that populate these tables from mainframe downloads and so forth). In other shops, the DBA is given the application by developers or the application is a commercial product. In some shops, object design comments by the DBA are appreciated. You need to be sensitive about your local culture and do the best you can to ensure that you have a database that is not a maintenance nightmare.

Much of the soundness of a given application is determined by following a few simple principles. (Chapter 39 discusses sound object design.) However, here are a few highlights to make this discussion flow more smoothly:

- ◆ A normalized table design, wherein the data is divided into a series of tables, each containing one conceptual type of data, is consistent with Oracle's design philosophy. A properly normalized database will usually perform much better under Oracle than one that is not normalized. You should review this during the design process to ensure that you have an application that can be tuned to meeting response time criteria.
- ◆ In a normalized table scheme, the tables are connected together through a series of common data fields known as *keys*. If your key structure is unsound, your problems may range from poor performance to incorrect answers. Therefore, focus on the keys during the design process.
- ◆ For decision support systems, you may wish to denormalize the database somewhat to improve speed and make access to information easier for quick-access tool users.

The design of the tables and their contents is often referred to as the *logical design*. It shows the data groupings and how they are related, regardless of how they are actually stored. In most of the instances that I have worked on, the DBA is responsible for the other half of the design, which is commonly referred to as the *physical design*. The physical design process in Oracle maps tables and indexes to tablespaces. Because tablespaces map to data files, which map to disk drives, this process places objects physically onto your various disk storage devices. Note that certain development methodologies have slightly different definitions for these steps. However, for now, understand what steps are involved in completing the table design. Figure 28.2 illustrates the differences between the physical and logical designs.

Figure 28.2.
The physical and
logical designs.



There are a number of special considerations that you need to take into account when you are completing the physical design process. This is especially important as a DBA because very few developers have enough training in operating systems, system architectures, and the internals of Oracle to perform this function. As with so many of the topics in this book, there are many considerations that can come into effect in certain circumstances. However, the following tend to be the ones that most commonly are considered when performing the physical design:

- ♦ What disk resources are available to you? This is the starting point for all other analyses. If you do not have an exact figure because the system is still being purchased, estimate the number and size of the disk drives that will be available to you. You can try other scenarios later, but sometimes you can learn a lot by just jumping into the process.
- ♦ You need to balance the transaction input/output load between disk drives. Because each transaction causes a write to the online redo log files, the rollback segments, the data files, and the indexes, you need to split these files and tables onto different disk drives whenever possible. This is complicated by the fact that you may have a large number of tables, tablespaces, or even applications competing for transaction input/output capacity. Always start with the most commonly accessed tables first and then work

your way down to the infrequently used ones. If you cannot find a good solution for the rarely used tables, it will not cause you much of a problem.

- ◆ For queries, you also need to balance the input/output load between disk drives. Most queries hit both the tables and at least one of the indexes associated with those tables. In addition, if you do a lot of sorting, you will read from and write to the temporary tablespace. As with transaction loads, start with the big and most frequently accessed tables first and then work your way down to the little and infrequently used tables.
- ◆ In addition to splitting the tables and their indexes, you may wish to split tables between various tablespaces and disks to balance the input/output load.
- ◆ You should also consider the queries that are common to the application when splitting tables between disks. If two tables are almost always accessed at the same time, performance will typically increase if they are located on separate disks.
- ◆ If you have a small table that is used frequently, you may want to consider loading this table into memory. Obviously, memory is very limited compared with disk resources. This can be a useful trick, however, especially when you have a data warehouse wherein the queries retrieve an enormous number of rows and the likelihood of a given row staying in memory until the next query is small.
- ◆ If you have a fairly large Oracle instance, you may want to consider more than just the disk drives when balancing input/output load. Disk drives are attached to controllers. On systems with multiple controllers, you may wish to locate disks that are accessed at the same time on different controllers to balance the load between the controllers.
- ◆ If at all possible, locate the online redo log files and their associated archive log files on separate disk drives. This prevents the system from having to read from and write to a single disk drive at the same time when copying the online redo log file to an archive log file.
- ◆ Finally, and this is a fairly complex topic that comes up on large data warehouses, you may wish to consider using disks with different capabilities for different purposes. Users typically think of disk drives as just disk drives. However, some drives specialize in high-speed access and others provide inexpensive bulk storage. There are special devices such as RAM disks that are extremely fast and other devices such as CD-ROM drives that are very inexpensive for mass data distribution. Finally, you can usually take disk drives and stripe them together to balance the load among multiple disk drives. If you have a complex instance where performance is an issue, talk to your system administrator or hardware folks to see whether they have any good ideas for you in this area.

Tip

Splitting tables and their indexes on separate disk drives is the first item to consider when performing the physical design and usually gives the greatest single improvement in performance.

You may have guessed from the previous discussion that I have spent a lot of time trying to maximize the performance of large Oracle instances. If you have a small database on a small server, you will probably not get involved with much more than locating tables and indexes on separate disk drives. However, if you have the pleasure of dealing with large instances, these and other factors will become important to you in the physical design process. One final warning: this is not a process that is completely analytical. You may look at the logical design and make guesses based on your experience as to the best configuration of the instance. You should monitor the instance once it is in production to see where the bottlenecks are and work on ways to improve performance from there. With any system that has such a large number of variables to control, trial and error can be very useful in getting that last ten percent of performance out of the system.

Before you complete your physical design, it is important to consider the indexes that will be used in your system. Again, it varies between computer shops as to whether the developers are solely responsible for designing indexes or whether they do not have a clue as to what an index is (perhaps that's too harsh, but you get the general idea). Not all storage systems use the concept of indexes, and therefore many developers may not have run across them. When you select one or more columns for the index, Oracle copies all the values from the table for these columns into the index and provides a link to the appropriate row in the base table. Indexes are stored in order and Oracle uses a very efficient search algorithm that has good performance even as the table grows much larger. If you have a fair-sized table that is commonly accessed based on a few key values to return only a few rows, an index is a good idea.

Table design can be fixed by going through existing paper forms and data that the users indicate they need in an application. However, indexes are driven by a knowledge of what is in the tables and how it is accessed. A common use of indexes is one in which a primary key must be unique (no two individuals can have the same social security number in the United States, for example). A unique index enforces this uniqueness and also gives you speed when you are looking up a person. Other indexes need to be determined based on the queries that the developers expect to be writing. You need to look at the `select` statements that they will be using and look at the `where` clauses to figure out what they are searching for. If they always need to retrieve the entire table (to perform a summation or something like that), indexes are not needed. However, if you are always going to scan entire tables, be ready for much slower response times as the tables grow in size.

There are a few points to remember about indexes. First, they are sensitive to the order in which you list their components. Therefore, when writing queries, make sure that you list the `where` items in the order that they appear in the index. Second, remember that you are balancing query efficiency against transaction efficiency. Although indexes save a lot of time on queries, they need to be updated on each transaction. This takes processing time to figure out where to place the value within the index and input/output bandwidth to transfer the data. It is not usually a problem unless you try to create an index for every field in the table. Third, you may want to occasionally re-create indexes on tables where you delete data routinely. Index records are not deleted, but marked inactive. This can result in having to retrieve a number of unnecessary rows. Finally, you have no control over the use of indexes. Oracle automatically determines whether it will use an index based on the fields you use in your `select` statement's `where` clause. If you are using the cost-based optimizer, the system will also factor in statistics about the tables involved. All indexes for a given table are updated on each transaction made to the system. It is not like dBASE wherein you have to use the `re-index` statement to rebuild the index after a number of updates.

Now that you have a fine set of indexes that will make your application incredibly fast, it is time to move on to the next topic required in the physical design of the database—sizing. The last chapter presents several methods that you can use to estimate the size of a table based on expectations of future growth. These all focus on the tables themselves. When sizing indexes, you can use the same methods for percentage increase or analogy with an existing system. When calculating size based on average row size, you need to modify your `select` statement, which determines the average row size to include only those fields that are within the index.

Now for a little refinement: You may have wondered about the accuracy of the row sizing estimation routine discussed in the last section. Some of you may be familiar with the storage parameters, such as those used to indicate percent used and percent free, which leave space for future activity in each row. It also is reasonable to assume that there is some overhead storage space required in a table. You're right. The exact method for determining disk storage includes these and other factors to arrive at an exact number.

I once went through the Oracle documentation and found its methodology for calculating disk storage space precisely. I then rearranged the process and adapted it to a spreadsheet that I use to calculate table and index sizes exactly (because I have no desire to spend a lot of time with calculators and pieces of paper). I could go through the nauseating details of the various storage parameters, but it is easier just to tell you that a copy of that spreadsheet is included on the disk for you to use. If you want to go through the details of how Oracle derived these formulas, look in your Server Administrator Guide under "Managing Schema Objects."

For now, here is a copy of this spreadsheet for your consideration. It is divided into two parts—one for tables and one for indexes. This is necessary because they have radically different overhead storage needs. Each of the parts of this spreadsheet has a general input section at the top wherein you input some of the basic database storage and tuning parameters that will be set up in your `init.ora` file (or you accept the defaults for your operating system). Once you fill these in, list your tables and indexes down the left side. You need to run the row counts and average size-per-row queries as discussed in the last chapter, which should be similar to the following examples:

```
SQL> select count(*) from orders;
```

```
COUNT(*)
-----
      61
```

```
SQL> select avg(nvl(vsize(customer),0))+avg(nvl(vsize(date_ordered),0))+
2  avg(nvl(vsize(amount),0))+avg(nvl(vsize(date_shipped),0)) "Avg Size"
3  from orders;
```

```
Avg Size
-----
      22
```

The spreadsheet is called `sizer.xls` and `sizer.wk3` (Excel and Lotus formats, respectively). You fill in the blanks as best you can and the spreadsheet goes through a series of calculations to determine the answer (remember that it is just an estimate and therefore you have to live with some errors being possible and beyond your control). Many of the columns that form intermediate results are hidden so that you have something that can be printed out and contains only the relevant information. A sample of this for the tables discussed in the last chapter is presented as Figure 28.3.

Let's look at one more topic on the design process, and then you can actually begin to implement some of these ideas. This topic pertains mostly to large data warehouses. In these instances, you may find that the basic data tables are so large that you cannot get adequate response time for common queries. If these queries are performing a repetitive series of calculations (for example, total sales by store or product line), you can improve performance by building summary tables that calculate these values in advance. When the user wants the data, it is a simple retrieval of a few rows from the summary table as opposed to a complex calculation from the base data tables. It is important when you are completing your design analysis to decide on whether you need these tables in your instance. It is also helpful to locate the summary tables on separate disks from the base tables because you will be reading from the base tables in order to write to the summary tables.

Figure 28.3.
Sample table and index
size calculations.

Table Space Calculation Sample

Givens for the Instance:

Fixed_header	57
Table_directory	4
Block_size	4096
Row_header	3

Calculations:

Table	PCTFREE	INITRANS	# Rows	Columns <250 B	Columns >250 B	Avg Row Size	Blocks	Bytes
customers	10	1	35,000,000	2	0	18	243,056	995,557,376
orders	10	1	61	4	0	22	1	4,096
	10	1			0		0	0

Table Sum 995,561,472

Space for Indexes Calculations

Givens for the Instance:

Fixed_header	113
Block_size	4096
Entry_header	2
Rowid_length	6

Calculations:

Index	INITRANS	PCTFREE	# Rows	Columns <250 B	Columns >250 B	Est. Avg. Data Space	Blocks	Bytes
customers_idx	2	10	35,000,000	1	0	5	145,257	594,972,672
orders_idx	2	10	61	2	0	9	1	4,096
	2	10	0		0		0	0

Index Sum 594,976,768

The actual command to create a table is relatively simple to deal with in its basic form (Chapter 39 discusses how to determine some of these parameters in more detail):

```
create table table_name
(column1      type(size),
 column 2    type(size),
 ...
 column n    type(size))
initrans #
maxtrans #
tablespace tablespace_name
storage(initial #
          next #
          pctincrease #
          minextents #
          maxextents #)
```

The column names and types/sizes are determined by the developers who analyze the data requirements. The key parameters affecting the DBA that are discussed in this section are tablespace, initial, next, and pctincrease. The tablespace line specifies the name of the tablespace in which this table will be created. This is how you move tables to where you want them. The initial and next parameters specify the initial and next extent sizes. Typically, you want to specify the initial extent to be large

enough to hold all the data in a single extent. The next extent parameter should be sized based on the size of the initial extent to accommodate overflow if you get more rows than you expected. For example, you would not pick a 4K next extent size on a table that occupies 1.5G because a small percent error in your size calculation would fill up a large number of additional extents if they were sized at only 4K. The final parameter is `pctincrease`. This factor is applied to all additional extents after the second, enabling them to grow larger and larger. I usually set this to zero because I like to know exactly what my extent sizes are. Here is an example of a `create table` command:

```
create table customers
(customer varchar2(5),
sales_rep varchar2(15))
initrans 1
maxtrans 100
tablespace user_data
storage (initial 4K
        next 4K
        pctincrease 0
        minextents 1
        maxextents 100)
```

The format of the command used to create indexes is somewhat similar. Here, you need to specify only the name of the table and the names of the columns. Oracle figures out the sizes of the columns from the way they are defined in the base table. The following is the format of the `create index` command:

```
create index index_name
on table table_name
(column1,
column2,
...
columnn)
initrans #
maxtrans #
tablespace tablespace_name
storage(initial #
        next #
        pctincrease #
        minextents #
        maxextents #)
```

A few notes are in order related to index creation. First, the `initrans` parameter has to be at least 2 for an index. (I never bothered to ask why, I just accept it because I cannot change it.) Second, the owner of the index does not have to be the owner of the table. This may seem strange, but if you are granted the index object privilege on the table or have the `CREATE ANY INDEX` system privilege, you can do it. Finally, the key to controlling where the index physically resides is the `tablespace` parameter, just as it is for tables. Here is an example:

```
create index customers_idx
on customers (customer_name)
```

```
initrans 2
maxtrans 100
tablespace user_index
storage (initial 4K
        next 4K
        pctincrease 0
        minextents 1
        maxextents 100)
```

One final tip related to the creation of tables and indexes. These objects are going to change over time. You may wind up having to move them between tablespaces or add a field to them. Changing the size of the tables as the instance occurs is a common problem. Finally, you usually will create tables and indexes in a test instance and then have to create them again in the production instance during application transition. A useful technique in these cases is to store all the object creation commands (views included) into SQL scripts on disk.

This technique enables you to make a quick change later on a rerun of the script. It also allows you to proof your spelling and other typographical features while still in test and have a reasonable assurance that the table will be created correctly when you have to go to production (which is usually a busy time period). You can use whatever text editor that you prefer and type in the series of SQL commands just as you would at the SQL*Plus prompt. Here are a few suggestions for building these scripts:

- ◆ Include a drop command before your create commands. If the object does not exist and you drop it, it gives you a minor warning and continues on. If the object does exist and you do not drop it, you get an error that prevents your create from happening. If you have a lot of tables with similar names or you work in a number of schemas, you may chose to make the dropping of tables a manual step to minimize errors.
- ◆ Include the create statement next, as discussed previously.
- ◆ For tables and views, use public synonyms to prevent having to type the owner name, a period, and then the table name in every application and query. Drop the public synonym before creating it, following the same logic presented previously for the creation of the table or index.
- ◆ For tables and views, try to include a brief comment on the table. This provides some basic online documentation that can be accessed through SQL queries (for those of you who can never find a particular document on your desks).
- ◆ Finally, try to include some brief comments on each of the columns in the table that describe the purpose of the column and some of its restrictions (for example, Y=yes and N=no).

Here is an example of a script that you might use to create the now-famous customers table:

```
drop table customers;

create table customers
(customer varchar2(5),
sales_rep varchar2(15))
initrans 1
maxtrans 100
tablespace user_data
storage (initial 4K
        next 4K
        pctincrease 0
        minextents 1
        maxextents 100);

drop public synonym customers;

create public synonym customers;

comment on table customers
  is 'Table showing list of valid customers & their sales reps';

comment on column customers.customer
  is 'Standard abbreviation for the customer';
comment on column customers.sales_rep
  is 'Name of the assigned sales representative';
```

MONITORING TABLES AND INDEXES

You really do not have to monitor your tables and indexes. You can just sit around and wait until a user gets an error message saying that a table has exceeded the maximum number of extents or your nightly batch processing routine bombs in the middle of the night due to some problem with one of the indexes. It is a very gutsy approach to live life on the edge and face danger at every turn. I, however, always implement a monitoring program to try to detect situations that might lead to problems and correct these situations before any of my users or developers know about them.

It is not that difficult to implement a monitoring program. Most of the basic tools are already in place for you. Part 4 of this book covers the basics of automation scripts and routine maintenance plans. The only thing left to determine is the parameters that need to be monitored. This can be a challenge within Oracle. I have seen third-party applications designed to monitor parameters in Oracle. They have screen after screen of graphs and numbers that take forever to wade through. Oracle also has many views, each of which has a number of parameters. Some of this data is useful for routine monitoring, but most is not.

Therefore, let's stick to basics. If you are dealing with table and index monitoring, there are two basic parameters that you need to track. The first is the total size of the table. Tables may grow over time beyond your forecasts. These changes may well affect your carefully laid plans to balance input/output loading between disks. It may also forecast a problem of running out of space in a given tablespace.

The second statistic to monitor for tables and indexes is the number of extents that they are occupying. This goes back to the fragmentation discussions in the last chapter (there also is a discussion of when to worry about fragmentation later in this chapter). For now, you need to monitor fragmentation to ensure that you do not exceed the number of extents specified in the `maxextents` storage parameter. If you do exceed that number, your transaction will fail with a nasty error message and everyone will come looking for the DBA. There is a maximum number of extents for the database, which is determined by the operating system and block sizes used.

The disk included with this book includes a copy of the utilization report that I like to run on most of my instances. There are a few instances that contain so many tables (for example, Oracle Financials instances) that I take out the individual table parameter queries and instead list only those that are becoming fragmented. This prevents me from having to deal with hundred-page reports. Here are a few of the queries that I use in this report, illustrating the basic monitoring concepts as they relate to tables, indexes, and other objects:

```
prompt
prompt Table Sizes Report
prompt

select tablespace_name,segment_name,owner,sum(bytes) Bytes
  from sys.dba_extents
 where owner not in ('SYS','SYSTEM') and segment_type='TABLE'
 group by tablespace_name,owner,segment_name
 order by tablespace_name,bytes desc
/
prompt
prompt Index Sizes Report
prompt

select tablespace_name,segment_name,owner,sum(bytes) Bytes
  from sys.dba_extents
 where owner not in ('SYS','SYSTEM') and segment_type='INDEX'
 group by tablespace_name,owner,segment_name
 order by tablespace_name,bytes desc
/
prompt
prompt Views Report
prompt

Select owner,view_name
  from sys.dba_views
 where owner not in ('SYS','SYSTEM','PUBLIC')
 order by owner,view_name
```

```
/
prompt
prompt Packages Report
prompt

select unique(name),owner
      from sys.dba_source
      where type='PACKAGE'
      and owner not in ('SYS','SYSTEM','PUBLIC')
      order by owner,name
/
prompt
prompt Table Fragmentation Report
prompt

select owner,segment_name,segment_type,sum(bytes),count(*) frags
      from sys.dba_extents
      where owner not in ('SYS','SYSTEM')
      having count(*) > 1
      group by owner,segment_name,segment_type
      order by frags desc
/
```

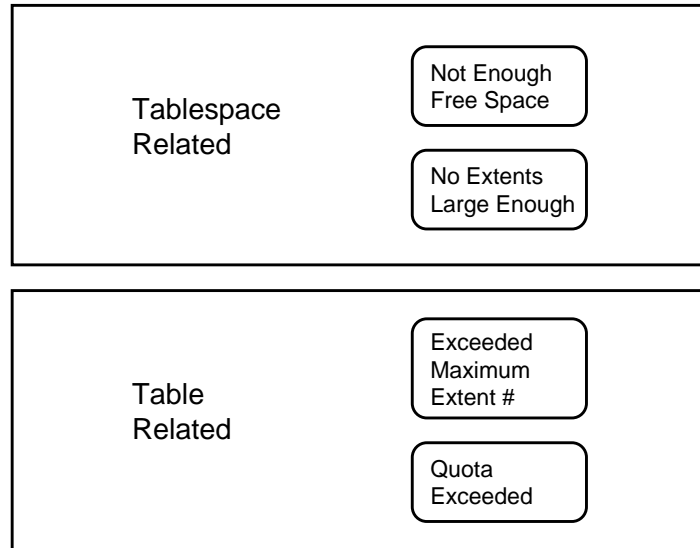
The basic principle of these is to make a query to one of the many views that Oracle provides the DBA. I have chosen these views because they represent the objects that I am most likely going to have to work with to support my users and developers. One feature that you should note is that I often put in a `where` clause that indicates that it should not report objects that are owned by `sys` and `system`. This saves me from having to scan through the long list of internal Oracle objects that do not typically have problems with growth. (Chapter 30 discusses the entire utilization checking script in more detail.)

One final thing to keep in mind when you think of monitoring your database is end user response time. This is the key parameter that will concern most of your users. They usually do not care about the internal technical details of the database or why things happen the way that they do. Rather, they are concerned only about how long it takes to get their jobs done. Therefore, it is a good idea if you check on response time, especially during busy periods, to see whether the growth in your tables has degraded performance to the point where it is time to take some action.

TYPICAL PROBLEMS AND THEIR SOLUTIONS

Problems with tables and tablespaces are often closely related to one another. It typically starts when people get a message that they cannot allocate space for additional data, their application bombs, and they come looking for you. Therefore, a good place to start is the drawing of the potential problem areas from the last chapter (see Figure 28.4).

Figure 28.4.
Potential problem areas
allocating extents.



The starting point for problem determination is the error message that you receive. If it mentions a specific problem, such as maximum number of extents reached, you can go straight to solving the problem. However, many applications do not pass the error messages through clearly, or users forget what the exact problems were. Therefore, you often have to start from scratch, and assuming that an application bombed due to storage problems in either the tablespace or tables is a good place to start. You can typically start by issuing the queries shown in the last section to determine the amount of free space and fragmentation for the tablespace as a whole. If you find that you have plenty of space and large extents available, proceed to search for any tables and indexes that have become excessively fragmented and reached the maximum number of extents allowed. Some queries to check this out are shown in the sample code in the previous section. Here is an example from a fairly healthy database:

```

1 select sum(total.bytes), sum(free.bytes)
2   from dba_data_files total, dba_free_space free
3  where total.tablespace_name = free.tablespace_name
4* group by total.tablespace_name
SQL> /
  
```

```

SUM(TOTAL.BYTES) SUM(FREE.BYTES)
-----
      3,145,728      2,734,080
    23,068,672      6,881,280
     2,097,152      2,095,104
     6,291,456      3,033,088
  
```

```
-- AND --
```



```

1 select segment_name,owner,count(*)
2 from dba_extents
3 where owner not in ('SYS','SYSTEM')
4* group by segment_name,owner
SQL> /

```

SEGMENT_NAME	OWNER	COUNT(*)
BONUS	SCOTT	1
CUSTOMER	SCOTT	2
CUSTOMERS	JGREENE	1
CUSTOMERS_IDX	JGREENE	1
CUSTOMER_PRIMARY_KEY	SCOTT	1
DEPT	SCOTT	1
DEPT_PRIMARY_KEY	SCOTT	1
DUMMY	SCOTT	1
EMP	SCOTT	1
EMP_PRIMARY_KEY	SCOTT	1
ITEM	SCOTT	1
ITEM_PRIMARY_KEY	SCOTT	1
ORD	SCOTT	1
ORDERS	JGREENE	2
ORD_PRIMARY_KEY	SCOTT	1
PERSONNEL	JGREENE	1
PERSONNEL_NAMES	JGREENE	1
PK_PERSONNEL	JGREENE	1
PRICE	SCOTT	1
PRICE_INDEX	SCOTT	1
PRODUCT	SCOTT	1
SEGMENT_NAME	OWNER	COUNT(*)
PRODUCT_PRIMARY_KEY	SCOTT	1
PRODUCT_PROFILE	JGREENE	1
REP_TEST_PARAMETERS	JGREENE	1
REP_TEST_RESULTS	JGREENE	1
REP_TEST_SAMPLES	JGREENE	1
SALGRADE	SCOTT	1
SCOTTTEST	SCOTT	1
USER_PROFILE	JGREENE	1

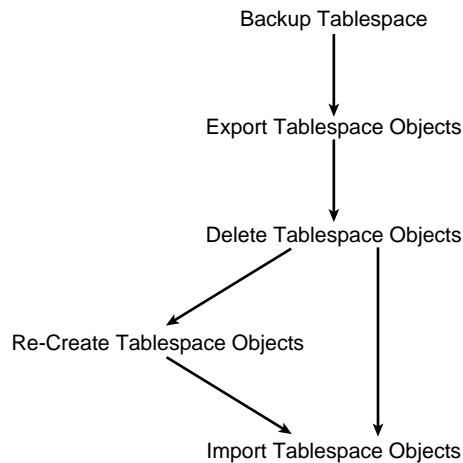
29 rows selected.

After running these queries, you should be able to determine whether you have any storage or fragmentation problems with your instance. In addition to these problems, there are others that may come up from the applications side. Examples of these include modification of fields, addition of fields, or even deletion of entire tables if they are no longer needed. The rest of this section covers how to accomplish the changes needed to solve each of these problems.

The first problem set that needs to be addressed is tables that are at or near (because you have a good monitoring program) their maximum number of extents. The solution to this fragmentation problem is similar to that used to fix fragmented tablespaces. This time, you export only the table or tables that are close to their

limits by selecting the `export table` option when prompted by Oracle Export. (I always do these tasks using the interactive mode of Export.) Have a good backup before you do any major surgery, in case you type something wrong or forget a step because of a telephone call in the middle of the process. If you remember to choose the `compress extents` option, you can merely delete the table and re-import it. It will now occupy a single extent. However, if you have used scripts to build your tables, you have the option of figuring out the size that you expect this table to grow to and then modifying the extent size parameters in your creation script. If you choose this option, you merely run the creation script after the export is completed (because you put a `drop table` command in this script) and then re-import the data. Remember to select `Yes` to the `ignore create errors` option on Oracle Import. Note that if you de-fragment individual tables, you have to be sensitive about the fragmentation of the overall tablespace. Remember, you will consume a single large extent and free up many small extents that may or may not be useful to other database objects. This process is illustrated in Figure 28.5.

Figure 28.5.
De-fragmenting tables.



A related problem is that of indexes that have reached their maximum number of extents. This is much easier because the index is based on the data within the table. In this case, you merely drop the index (make sure that you have a creation script or know what columns and storage parameters are used in the index). Then re-create it with new storage parameters. If you used object creation scripts, you modify the storage parameters and run the script (assuming you put a `drop index` command at the beginning of your scripts). (With tables, I always worry a little even though I have never had problems with the export/import process—I am the nervous type.) Index resizing is much less stressful because you are not messing with the underlying data. If you make a mistake during the index re-creation process, you just do it again.

Let's move on to a task that has nothing to do with data storage and Oracle internals. Over the course of the life of your applications, the developers may realize that they need to change the tables or indexes. Deletions are rare. Usually, they want to add more fields or more tables or make fields even larger. Perhaps it is all related to the information overload that everyone suffers. Oracle provides you with a simple solution for tables. All you have to do is issue the `alter table` command with either the `modify` or `add` option and list the columns that you want to change. If your change is severe (numbers to dates or something like that), you may have to make a copy of the table and then copy the data back after you have made the changes, using a script that performs the appropriate format conversions. But for now, let's look at a couple of simple examples. First, suppose you want to modify the customer field in the customers table to be a `varchar(10)` as opposed to a `varchar(5)`. You issue the following command:

```
alter table customers modify(customer varchar(10));
```

To add a new field—for example, a date to capture when the customer first orders a product from you—you use the following command:

```
alter table customers add(first_order date);
```

There may be times when you are concerned about the order of the fields within the table (either for readability when listing structure or for technical reasons). At other times, you may need to convert the format of some of your data fields. At these and other times, it might make sense to make a copy of the table, modify it, and then use a script to copy the data from the old table into the new table. This is not a complex process. First, you create a new table using the `as select` option. Then you perform your modifications. Finally, you use the `insert` statement wherein you select the values from the other table and perform some manipulations. For example, suppose that you want to modify the customers table to have the `sales_rep` field modified from `varchar(10)` to `varchar(20)` and you to remove rows where the salesman name is SMITH. To do this, you use something like the following:

```
SQL> create table cust_temp  
2 as select * from customers;
```

Table created.

```
SQL> alter table cust_temp  
2 modify (sales_rep varchar2(20));
```

Table altered.

```
SQL> truncate table customers;
```

Table truncated.

```
SQL> insert into customers
  2  select * from cust_temp
  3  where sales_rep not like '%SMITH%';
```

47 rows created.

Finally, it is a rare occurrence that anyone would want to get rid of data. People usually want to keep copies of data that they no longer use “just in case.” However, if you ever do get the opportunity, the command used to drop a table or index is relatively simple:

```
SQL> drop table cust_temp;
```

Table dropped.

```
SQL> drop index customers_idx;
```

Index dropped;

The answer to the question of index modifications is also relatively simple. If you want to add or remove columns from an index, you drop the existing index and create a new one that is set up the way you want. Again, this is a low-risk operation because you are not touching the fundamental data stored in the table on which the index is based. It is nice every now and again as DBA to have a task wherein you don't have to try and concentrate heavily while the phone is ringing off the hook.

One final note on routine modifications: If you want to change the tablespace or initial extent size of a given table or index, you have to export the table or drop the index and then re-create it with the parameters that you want. However, many of the other storage parameters such as those for next extent size or maximum number of extents can be modified using the `alter table` command. To do this, you merely use the `alter table` or `alter index` command and show the parameters that you want to change. As an example, suppose that you want to change the next extent size to 8K and the maximum number of extents to 50 on the customers table. Here is the command to accomplish this:

```
SQL> alter table customers
  2  storage (next 8K
  3  maxextents 50);
```

Table altered.

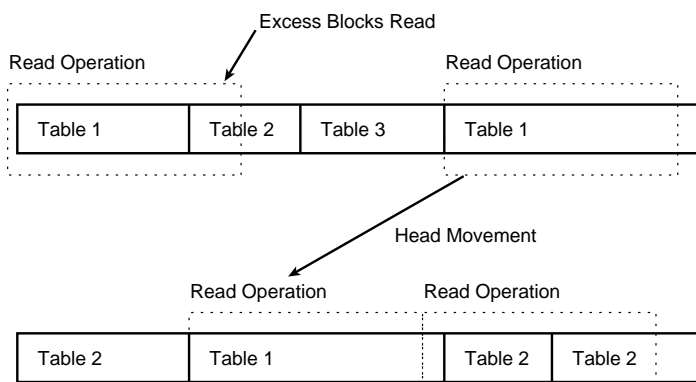
This section provides guidance on the routine problems that come up with tables and indexes. Some may not be considered problems in some locations. Perhaps they could be referred to as changes. There are usually several alternative commands that can get the job done. As you get more comfortable with the system, you may find that you do not like using table creation scripts or one of the other techniques presented here. That is okay. Every DBA has personal preferences and a unique work environment. You have to take the techniques presented here and either figure out which ones work best for your situation or adapt these to fit.

FRAGMENTATION

So far you have explored the case wherein you have severe fragmentation to the point where you have allocated the maximum number of extents possible (the `maxextents` parameter) and your applications will start bombing out the next time that you need to allocate an extent. Most people can understand this problem. However, what about larger systems on which the `maxextents` parameter can go to 256 or 512 extents? What are the impacts of having 300 extents on performance?

Recall that Oracle reads more than just the row that it needs. It reads multiple blocks of data in a given pass. Therefore, if your table is all in contiguous blocks, it will probably be able to pull all the data in one movement of the disk drive heads and be done with it. However, if your table exists in a large number of extents scattered across the disk, you have to position the heads and read from multiple groups of blocks to retrieve your data (see Figure 28.6). Disk drives are designed so that they read contiguous blocks relatively fast, but take longer to move disk heads and other such tasks. Therefore, if you have a large number of extents that are not contiguous (and may even exist on different disk drives), you have to read more bytes of information to get what you need and suffer performance hits related to disk drive head movement.

Figure 28.6.
Effect of fragmentation
on read operations.



Although I usually try to avoid fragmentation, there may be situations where it is not a problem. The following are some examples where it does not have a significant impact and it may actually be necessary:

- ♦ If a table or index is the only object in a given tablespace, it does not matter if you have multiple extents; they will be contiguous anyway.
- ♦ Because the largest extent that you can allocate is the size of an individual data file (which on UNIX is typically limited to 2G), you may need to have

multiple extents for very large tables. Tables that are this large usually have their own tablespace, and therefore the extents are going to be contiguous anyway.

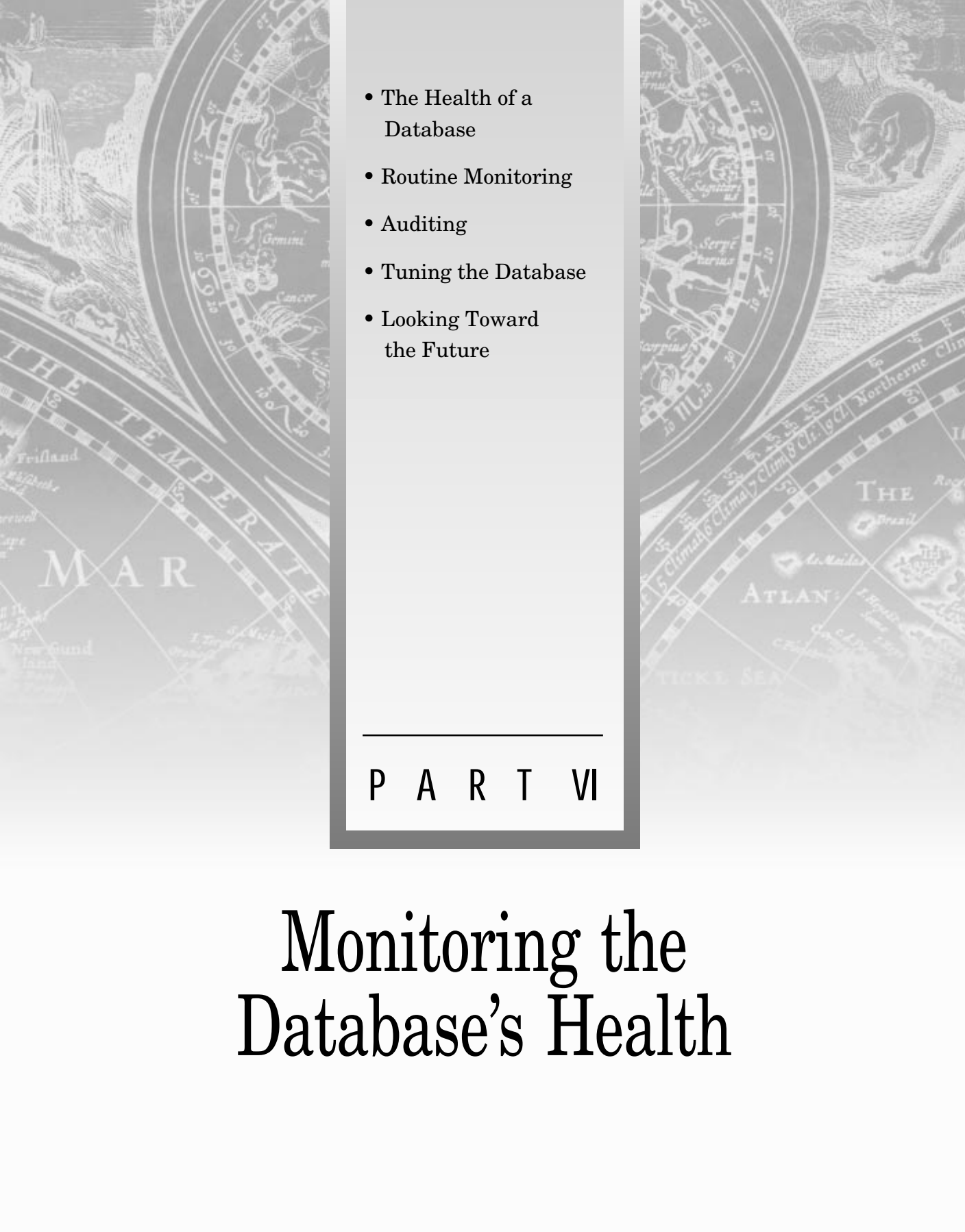
- ◆ If you are in an instance that is growing, but you cannot purchase all the disk drives that you will need in the future, you may have to let things grow slowly rather than waste disk space by allocating extents that are large enough to accommodate future data volumes. Think of this as living in the real world of budgets and fiscal periods.

SUMMARY

This chapter presented the routine operations that you will be asked to perform on the fundamental database objects—tables and indexes. Other objects, such as views, stored procedures, and synonyms, tend to be dropped or replaced and have little effect on overall data storage. You may run into strange situations and problems that are not covered in this chapter. Those issues are discussed in Part 7, which deals with problems.

This chapter is also the conclusion of the part of the book that has been devoted to the daily routine. In the buildup of knowledge so far, you started with some technical fundamentals that you need to grasp before you can understand what is happening. Then you learned about the installation and upgrade of the RDBMS software itself. Next you explored advanced planning for the DBA job in a database administration scheme. This part of the book covered common tasks with which you should feel fairly comfortable.

The next part of the book discusses monitoring in more detail. Basic checks are presented while discussing the daily routine. It is time to build on these basics so that you can feel comfortable with your ability to see what is really happening in the database. After you explore monitoring, you learn how to deal with problems and support users and some advanced (more technical) features of Oracle that you might want to consider.

- 
- The background of the slide is a faded, historical map of the Atlantic Ocean. The map includes zodiac signs such as Gemini, Cancer, and Sagittarius, and labels for various geographical features like 'THE TEMPERATE MAR', 'ATLAN: TICKE SEA', and 'Brazil'. A central white box with a grey border contains a bulleted list of topics.
- The Health of a Database
 - Routine Monitoring
 - Auditing
 - Tuning the Database
 - Looking Toward the Future

P A R T V I

Monitoring the Database's Health



- What Is a “Healthy” Database?
- Monitoring Programs
- Auditing
- Tuning

CHAPTER 29



The Health of a Database

As you have learned, I have spent a lot of time devising monitoring routines and scripts to try and avoid problems. This is necessary as Oracle databases move from labs and small operations to support major production systems. This world is used to extremely high standards of availability and is willing to pay for the time and resources that it takes to meet these reliability goals.

This chapter begins with an overview of what makes a healthy database. If you set your goals up front, you know what to look for along the way. Part of the term “healthy” relates to the reliability issues previously discussed. Although this is an important part of a healthy database, it is not the only part. Included with reliability is performance that meets user requirements and security that comes from having data properly protected. Taken together, these components provide a sound foundation on which applications can be built.

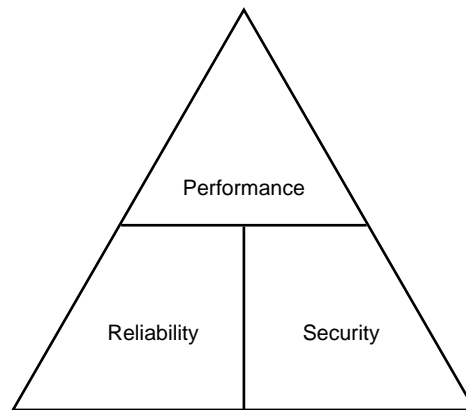
This chapter on database health serves as the basis for the monitoring routines that are discussed in Chapter 30. Monitoring serves as a verification of the health of the database. It is also a necessary component to ensure the reliability of the system. After you learn about monitoring, you will explore auditing, which is a related concept. Auditing is a record of activities that are being performed within the database. It typically has a security flavor to it, but it can also show you how the database is being used. Once the monitoring and auditing discussions are completed, it will be time to discuss the important performance topic of database tuning. The goal of tuning is to ensure that the database is ready to accept the loads placed on it, limited only to its host computer system and the applications that access it. The final chapter in this part of the book deals with future planning. As you may have guessed from the previous chapters in the book, this type of planning is one of my favorite topics. This discussion is focused toward planning that can enable you to make the database healthier in the future.

This chapter begins with a definition of a healthy database. Then you will look at an overview of monitoring programs that will be discussed in more detail in Chapter 30. From there you move to an overview of auditing as an introduction to the more detailed discussions in Chapter 31. Finally, you explore the basic concepts of tuning, which is perhaps the most complex topic in this part of the book.

WHAT IS A “HEALTHY” DATABASE?

Recall that the concepts of reliability, performance, and security are part of what I call a “healthy” database (see Figure 29.1). Actually, although I do not see much of this concept in the computer field, the health of a database is a measurement of when the database is doing the best that it can. In line with the philosophy that you cannot manage something unless you can measure it, let’s define the standards of “normal” for an Oracle database.

Figure 29.1.
Components of a
healthy database.



This is not a philosophical debate in most computer shops, where all too often you find a lack of understanding about how this new UNIX operating system, the new RISC processors, the new RDBMS, and the new application development tools work together to provide an application for the users. In any situation where things are confused, there is a tendency to point the finger at those parts of the system that are least understood when things go wrong. The Oracle RDBMS is a likely candidate for such finger pointing in most organizations. True, most people understand the basic concepts of what the database is designed to provide for the organization. It stores data in a retrievable fashion and provides a number of other data-related services. However, the internals of the system are a mystery understood by very few people. Therefore, when an application is too slow, there is a tendency to consider that the database is too slow rather than that the query is poorly formed. When the database crashes due to a lack of space, it is the database's fault.

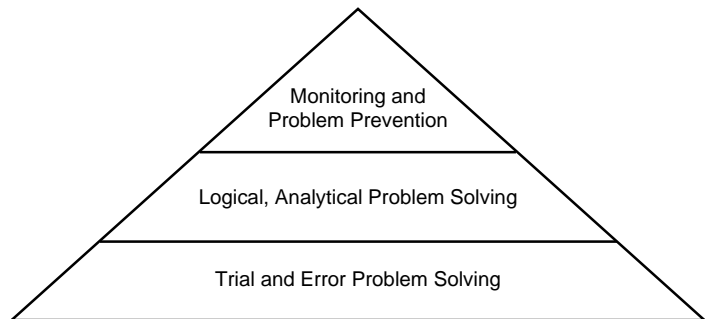
I've been in all too many meetings where there were finger pointing and unhappy people. Computer personnel are under a lot of pressure these days to be perfect, cost-effective, and totally responsive—or their jobs will be out-sourced. They all argue that the problems are not within their areas of responsibility, but too few present any technical evidence to back up their claims. Your goal in a monitoring and auditing program is to measure the performance of your RDBMS so that you can track its performance and compare it against accepted standards.

Let's focus on accepted standards of performance. In a room where all claim that their parts of the system are working just fine, it often comes down to seniority or the loudest shouting voice to determine who is "right." Your opinion will not be the basis for the discussion on the database; nor will mine. It's a good idea to check the large volume of Oracle literature to determine the standards that Oracle provides on the subject. Associated with each standard is a set of actions that can be taken to correct values that exceed the standards. Some of these actions will be within your

capability (for example, adjusting some tuning parameters) and others will not (for example, increasing system memory or computing capacity). The key here is that you have a process that is approved by the experts at Oracle, and very few people in your organization will be in a position to argue Oracle internals with Oracle.

The Software Engineering Institute at Carnegie Mellon University publishes a software development maturity model used to describe how organizations approach the software development process. The model looks at your practices and puts you on a scale that ranges from shops that just assemble software on the fly to those shops that have a very mature, repeatable, and reliable development process. Consider where you want to be on a scale similar to this when it comes to database administration (see Figure 29.2). On one level, you can ignore the system until a problem comes up and then use trial and error to fix the problem. The next level might be that you have a process in place to figure out logically what is wrong when the system breaks down. The highest level, and one I recommend that you consider, is one where you are continuously monitoring the health of your database and taking action before problems arise.

Figure 29.2.
A database administration maturity model.



The first component of a healthy database is reliability. There are many components of reliability that are beyond your control. For example, Oracle's capability of retrieving the correct rows associated with a given query are controlled by the RDBMS itself. Bugs that cause the RDBMS to crash or not perform correctly are also beyond your control. However, the RDBMS software itself is pretty reliable. The majority of the problems that can occur are within the control of the DBA. You just need to know what is likely to occur and keep an eye out for these potential problems. The monitoring routines presented in the next section are based on the experience gained and published by Oracle combined with some of my own experiences. This should catch most of the common problems that are within your control.

The next component of a healthy database is performance. "More for less" are the buzzwords at many corporations these days, and you are probably being asked to squeeze as much as possible out of your computer systems. Gone are the days where

multimillion-dollar mainframes were being purchased routinely just to keep reserve capacity available. (I once saw an entire mainframe whose purpose was to support special testing and provide a hot backup computer system.) Now you are being asked to take that little UNIX box and make it do what the mainframe used to do. Worse yet, the applications are becoming increasingly complex to provide better service to the users. Therefore, your job is to look at your Oracle instance and see whether it is possible to increase its performance. The tuning report discussed as part of the monitoring program looks at the key performance tuning parameters. When combined with the tuning discussions, it will enable you to get as much as is reasonably possible out of your system.

Security is also an essential unit in the concept of a healthy database. There are some locations where everything is open and security is not a concern. However, most of the places in which I have worked on Oracle need some level of security to protect the data that is stored within the database. It does not have to be top-secret, national-security information, either. Personnel records can be extremely sensitive and there is an enormous temptation to adjust financial records when large amounts of money are involved. This is perhaps the most difficult section in which to assess your standards of health because it requires value judgments as to what is “good enough” for security. Later, you will study a few ideas for standards in this area and also look over a report that enables you routinely to check whether your security scheme is still working correctly.

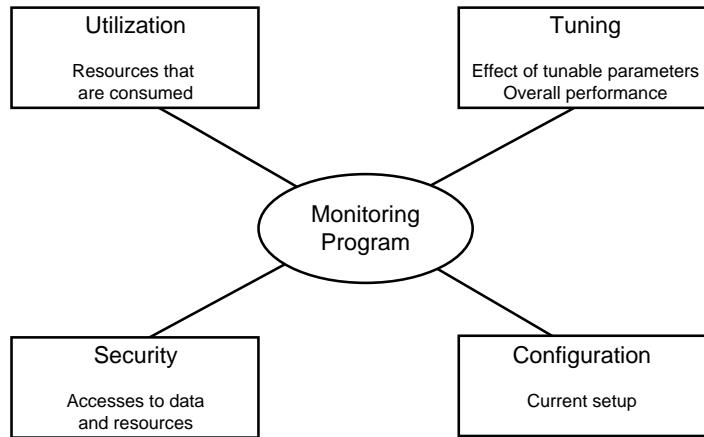
I run four standard monitoring programs, which are provided on the enclosed disk. Each of these reports focuses on one or more of the health components that I have discussed in this section.

MONITORING PROGRAMS

Most DBAs suffer from a common problem in the world today—information overload. (Some might say we cause it.) Therefore, a routine monitoring program needs to be a balancing act between getting a complete picture of areas that can cause problems and minimizing the volume of material that you have to read. The program that I have implemented draws information from a number of different sources, including Oracle documentation, performance tuning books, and bulletin boards. After this research, I formed a number of SQL queries that look at Oracle to determine its status. Because I like to have things organized, I have divided these queries into four distinct reports that I run on a regular basis (see Figure 29.3):

- ◆ Resource Utilization
- ◆ Tuning
- ◆ Security
- ◆ Database and Instance Configuration

Figure 29.3.
Routine monitoring reports.



I run the utilization report frequently (once or twice a week). The goal of this report is to monitor those resources that are consumed over time, especially data storage space. The following are the items included in this report:

- ♦ Tablespace utilization (bytes consumed and bytes free)
- ♦ Data files associated with the tablespaces
- ♦ Nonsystem tables and their sizes in bytes
- ♦ Nonsystem indexes and their sizes in bytes
- ♦ Nonsystem views
- ♦ Stored procedures
- ♦ Object fragmentation (count of the number of extents consumed by various objects)
- ♦ Tablespace free segments

The next report focuses on the tuning of the Oracle database. If you look at the internal statistics that Oracle gathers, you realize that you could fill page after page with statistical analysis. This report focuses on the parameters that typically need tuning in an Oracle instance, specifically in the UNIX environment. There are actually two components to this report. The first is a SQL script that monitors Oracle internal parameters in various Oracle environments. The other part of this report is some UNIX shell commands that call standard UNIX performance monitoring packages to determine the loading as seen from an operating system perspective. To adapt this to other environments, you replace these commands with those specific to your operating system (the monitoring utility under VMS). Taken together, this should give an accurate picture of the things that are most likely to require tuning:

- ◆ Library cache
- ◆ Data dictionary cache
- ◆ Multi-threaded server session memory
- ◆ Buffer cache
- ◆ Disk input/output
- ◆ RBS contention
- ◆ Latch contention
- ◆ Multi-threaded server dispatcher contention
- ◆ Shared server process contention
- ◆ Redo log buffer contention
- ◆ Sort memory contention
- ◆ Free list contention
- ◆ System computer loading
- ◆ System input/output loading
- ◆ System memory loading

The next report to be run routinely provides a picture of the security implementation within your Oracle database. This is a more difficult report to look through because, as opposed to measuring a database value against a given standard, you are looking at what accesses users have compared against your security scheme. Because it is more complex to analyze and tends to change less frequently than other monitored parameters, it should be run about once a month. After a while, you will know your security scheme and can review this report in less than five minutes. The data gathered by this report includes the following:

- ◆ Profiles
- ◆ System privileges
- ◆ Users
- ◆ Roles
- ◆ Role privileges
- ◆ Tables
- ◆ Table access privileges
- ◆ Views
- ◆ Synonyms
- ◆ Indexes
- ◆ Packages

The final monitoring report in this series is the configuration report. You do not need to run it very often—perhaps monthly. Its purpose is to capture the initialization parameters and file layouts associated with the instance at a given time. Although this does not have any effect on tuning and will not catch tables that are about to reach their maximum number of extents, this data can be useful when problems do arise. The question often comes up as to what has been changed recently that might have caused this problem to occur. You can use the configuration data on this report to search for possible causes. The data gathered on this report includes the following:

- ♦ Data files
- ♦ Redo log files
- ♦ Tablespaces
- ♦ Nondata objects
- ♦ Database objects by type
- ♦ Background processes
- ♦ SGA sizing
- ♦ SGA parameters

AUDITING

Conceptually auditing and monitoring are closely related. In both cases, you are trying to look at your database to see what is going on. The key difference is that monitoring tends to focus on measurable statistics on what the database has done as a whole, and auditing measures specific actions by end users. For example, monitoring captures the total activity to a particular disk, and auditing captures details about the times that someone used the `alter tablespace` command. It is also convenient to separate the two activities, because auditing is a special set of tables and utilities from those used to capture the information in the monitoring reports.

Oracle does not perform any auditing by default. You, as the DBA, have to turn on auditing. In addition, you need to specify the activities that you wish to keep track of as part of the auditing process. There are separate tables used to store the audited events that you need to collect data from and clean out routinely. This can turn out to be a fair amount of work. However, just like the monitoring routines, you can develop scripts that are executed automatically to provide reports from your audits and clean out the audit tables.

One of the key things you need to do before you activate auditing in your database is plan out what you need to audit. Oracle enables you to capture data on just about everything that happens in the database. Storing this information eats up a lot of

disk space and also slows down your processing activities because it takes time to record this audit data on each transaction or query.

TUNING

A logical consequence of the monitoring and auditing programs is the need to tune the database occasionally. Remember, Oracle gives you a lot of configurable parameters. The real question is which of these parameters to adjust in order to improve performance in your instance. The monitoring reports, specifically the tuning report, provide some fairly obvious inputs to the tuning process. You can gather useful data from other reports, such as the utilization report, to identify tables that are becoming large and may need to be relocated.

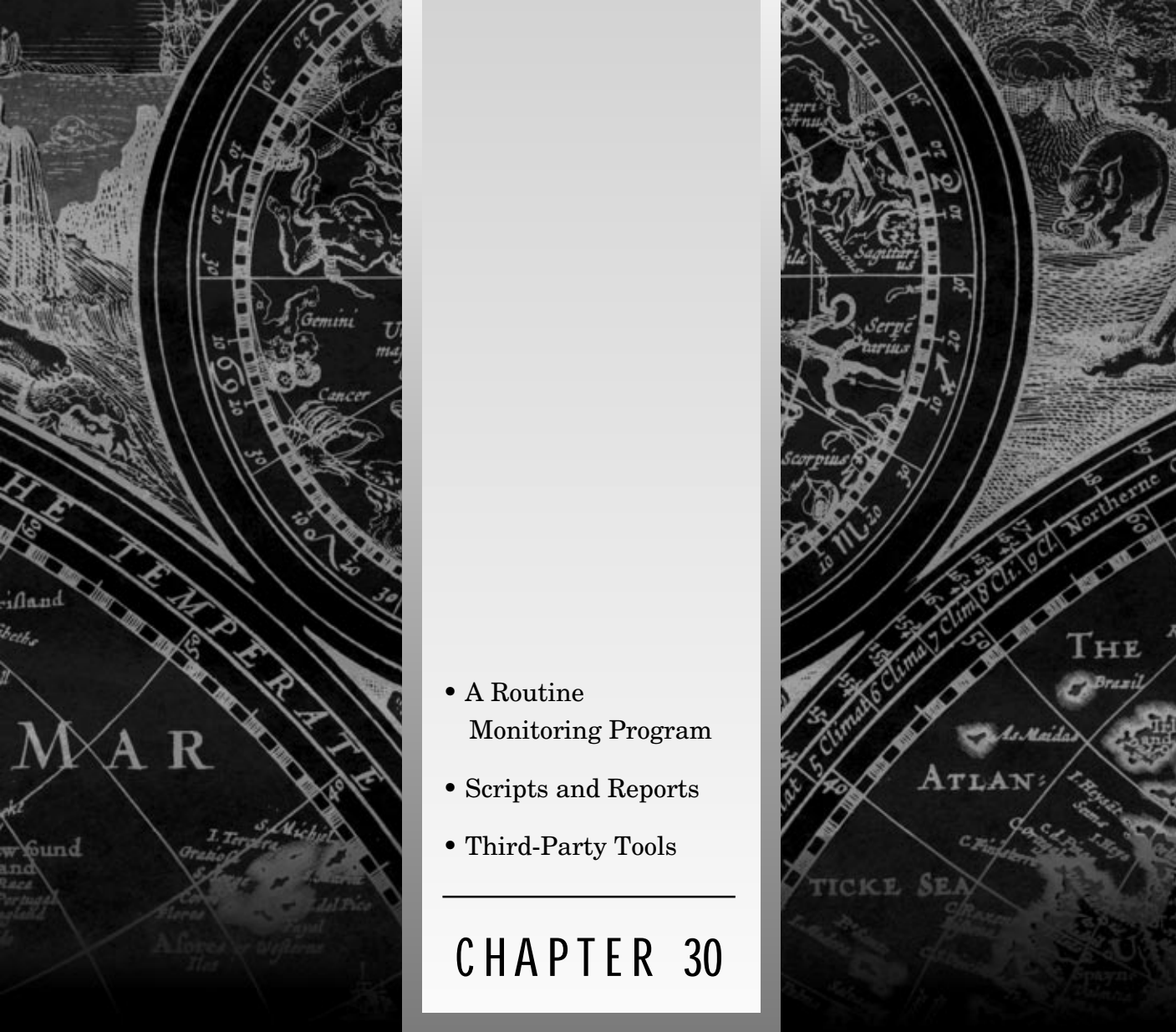
A less obvious alternative is to use the auditing utilities to provide inputs to the tuning process. The monitoring utilities capture the overall picture that something is happening and the auditing reports can capture individual events. For example, suppose that you notice that a particular disk is experiencing a lot of write activity. You have no idea whether this is due to a large amount of inserts, deletes, or updates. You are also uncertain as to which tables are causing this heavy activity. You have the option of turning on auditing of the insert, update, and delete events for the tables in this tablespace that you feel are likely to be causing the activity. You then can look at the auditing data to generate statistics about what is happening with each of these tables and perhaps move some of the tables or indexes to balance the loading, or generate new indexes if the update activity levels are high.

This brings up another important point. There are actually two components to tuning. The first involves adjusting database initialization parameters and object locations to provide the maximum performance from the instance itself. The second part of tuning involves designing the applications, specifically the transactions and queries, to achieve maximum performance. This may or may not fall under the job description of the DBA in your organization. However, you need to be aware of the techniques that can be used in order to point the developers in the right direction. Remember, you can have a perfectly tuned instance that can look really bad if developers write their queries poorly.

SUMMARY

This chapter was devoted to database health—how a database should be performing to support goals of reliability, security, and performance. It was designed around the concept that you have to be able to measure something (against an accepted set of standards) in order to be able to manage it. This may prevent endless arguments about the database being “too slow” or needing some (unspecified) improvements.

This chapter also provided a brief introduction to some of the topics that will be discussed in this part of the book. Think of it as a way of introducing the concepts and easing you into these subjects gradually. Each of these topics—monitoring, auditing, and tuning—has a chapter of its own. Your goal at this point is to understand the general concepts associated with each of these subjects, the differences between them, and how they work together to assist you in managing a healthy database.

- 
- A Routine Monitoring Program
 - Scripts and Reports
 - Third-Party Tools
- ## CHAPTER 30

CHAPTER 30

Routine Monitoring

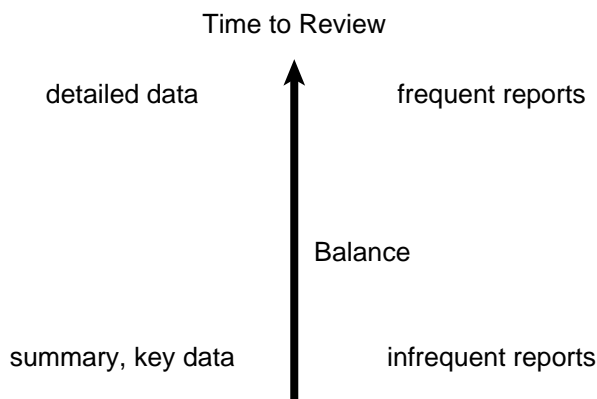
As introduced in the last chapter, the routine monitoring reports are a technique that you can use to determine the health of your database. In a reactive mode, you can use these reports to help determine where your problems are originating. In a proactive mode, you can run these reports on a regular basis to detect problems before they become crises. Most of these reports are designed to compare actual database values against accepted standards for the Oracle system. The others rely on your comparing the actual database configuration against your security scheme and other laid-out plans.

This chapter begins with a discussion of routine monitoring programs. Again, this is proactive problem avoidance. Next, you explore some general material on the scripts provided on the enclosed disk. Then you work through each of the four monitoring reports to show the various parameters monitored and the expected results. The chapter concludes with a few remarks about third-party products that can monitor Oracle databases.

A ROUTINE MONITORING PROGRAM

A routine monitoring program needs to balance a number of factors against one another. Obviously, you want to get a complete picture of what is happening in your Oracle instances. Murphy's Law states conclusively that if you forget to monitor just one of the many aspects of your Oracle instance, that will be the one that causes you problems. However, your time is limited and probably much in demand. Therefore, a monitoring program needs to track only those parameters that have a reasonable chance of causing problems (as opposed to the hundreds of possible parameters). You also need to adjust the frequency of the reports so that you get your information in time to take corrective action. The end result should be a program that neither overloads you with information nor keeps key data from you (see Figure 30.1).

Figure 30.1.
Balance of information
in a monitoring
program.



First, let's look at how to choose the parameters to monitor. When I first started building these reports, I started with a list of factors that seemed obvious—free space in the tablespaces, table fragmentation, and so forth. Then, over time, problems crept up that were not detected by the initial reports. One such example is the case wherein there was a fair amount of space left in a tablespace, but the individual free segments were too small to accommodate the extent size needed for a particular table. The solution to this was to add a query that listed the size of the free extents in the various tablespaces. A quick review of this output shows the lack of free segments with adequate size.

This brings up an important point to consider. Although there are a few absolutes—you need to keep track of the available free space in any tablespace in which you are adding data, for example—your individual situation may need some special attention. For example, if you are running an instance with Oracle Financials, you will have hundreds of tables owned by a half-dozen owners or more that are needed to get Oracle Financials going. In this case, you will find it impractical to list the size and number of extents for each individual table (the report can be over a half-inch thick). In a similar situation, I modified the report to list only those tables that occupied more than five extents. You need to look at the output of your reports and determine whether you need to add, delete, or modify the queries to be most effective for your needs.

The next task that you face when constructing a plan for monitoring your Oracle instances is to determine the frequency at which you are going to run these reports. It is true that over time you will become familiar with reading these reports and be able to focus in quickly on the numbers that are important. You will probably also come to know which parameters are potential problems for your various instances (for example, disk space in the test instance or fragmentation in your production instance). However, you do not want to run the full set of reports once an hour. This chapter features a schedule that I typically follow for the instances with which I work. You can adjust it to fit your needs.

What factors influence the frequency at which you run the various reports? First, I tend to keep a much closer eye on production instances than I do on test and development instances. Although developers have the capability of causing more problems than typical production users, the impact on the operation of the business is usually less severe if a problem occurs in the development instance (and the developers will usually go after their compatriot who just created a 2G table that filled up their development tablespace). Another factor to consider is the importance of the various aspects of database health to your organization. Some people care very little about security; in other organizations, it is the lifeblood. Finally, you need to adjust the schedule to fit the other things that are going on both in your schedule

and in your organization. For example, if there are very large batch updates that occur over weekends, it might be useful to run utilization reports on Fridays (to make sure that you are ready) and Mondays (to see whether everything went okay). You get the idea. This is where experience with what is going on in your world helps you predict and adjust your support to better meet user needs.

Let's look at a typical monitoring schedule. This is based on my experience, which has been more with large data warehouses that suffer from disk space and query performance problems. The security environment has focused more on routine business access control as opposed to processing extremely sensitive information. These also tended to be production environments wherein reliability was extremely important. The schedule looks something like this:

- ♦ Utilization reports run once per week on Monday morning (after large weekend batch updates).
- ♦ Tuning reports run once per week on Thursday morning (this gives sufficient time to gather representative statistics and plan any needed changes in time for the weekly Oracle instance restart on Sunday).
- ♦ Security reports run once per month on the fifteenth day of the month.
- ♦ Configuration reports run once per month on the first day of the month.

One final component that has been part of many of the systems that I have worked with, but often does not fall within the scope of the DBA, is a set of checks to ensure that the Oracle processes are still operational. You run a process status monitoring utility and check to see whether this list contains all the required Oracle background processes. For example, under UNIX you might check for the existence of the database writer process for the PROD instance with a command similar to this:

```
ps -ef | grep -v grep | grep PROD | grep DBWR
```

Such jobs are typically run on a fairly frequent basis—every five or ten minutes, for example. Typically, you send a message only to a console or a routinely monitored e-mail box when there is a problem. You should also consider implementing these monitors for related processes such as those used by SQL*Net. In many locations, these scripts are implemented by operations or system administrators to inform operators of problems. However, if you do not have this luxury, you may consider doing it yourself.

SCRIPTS AND REPORTS

It would be possible to take each of the queries in these reports and run them, one at a time, from the SQL*Plus command prompt. The typical work environment is too busy and the typing a little too tedious to attack the problem this way (computers

are such sticklers for syntax and typographical details). The logical solution is to store these queries in SQL program files and run them as needed. These can be organized into four separate reports:

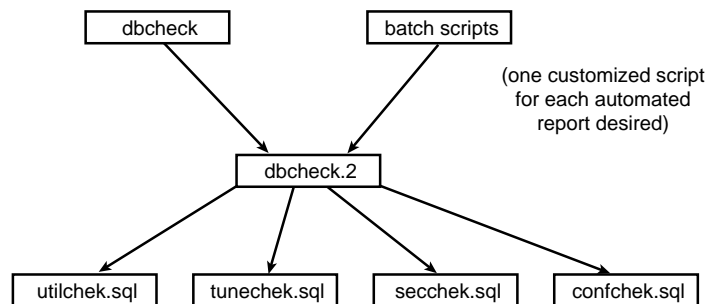
- ◆ Utilization
- ◆ Tuning
- ◆ Security
- ◆ Configuration

You may want to split certain queries or insert additional queries into these reports based on your personal preferences and experience. That is the great thing about learning script and shell programming. You can make the computer do what you want while you are off working on other things. Although it is possible to log into SQL*Plus and run each report individually and then move the output to where you want it, I have opted to build several UNIX shell scripts to handle this task for me. The accounts assume that the user has an OPS\$ account that has direct access to the DBA views (that is, I have run the catdbsyn script for this account).

I have used a series of scripts to make it easy to run these routines both from the command line and in a batch mode. The basic flow of these scripts is depicted in Figure 30.2. The dbcheck script is designed to get input from the user as to which instance is to be monitored and which report is to be run. The batch scripts have the name of the report, report destination, instance, and other necessary control information hard-coded.

Both the dbcheck script and the batch scripts call the dbcheck.2 module, which calls the appropriate SQL report script and takes care of mailing or printing the report.

Figure 30.2.
Basic flow of dbcheck
shell scripts.



The SQL scripts should work in any Oracle environment that supports SQL*Plus and Oracle 7. The dbcheck and batch scripts are designed for the UNIX environment. However, I have found that with the exception of the syntax details, most other shell scripting languages (DEC's DCL, for example) have the constructs

necessary to build scripts that perform this same function. One final note: the scripts have been designed to run under the Korn shell. If you do not normally use this shell, you can access it by typing `ksh` at the UNIX prompt.

UTILIZATION MONITORING

The first report is the utilization report. Its purpose is to capture disk space and storage utilization statistics for objects that are not owned by the Oracle system itself. This was designed to prevent you from having to wade through a large number of Oracle internal tables that do not typically have storage problems. This report monitors free space in the system tablespace, which is the typical problem that you run into for RDBMS internal objects.

At the beginning of all utilization reports, you will find a header that shows the instance name, date run, and other information that enables you look at a printout and understand where the data you are looking at has originated. The header looks like the following:

```
#####
#               Oracle Database Utilization Report               #
#####
```

Instance Name:

oracle

Date Of This Report:

30 May 1995 15:00

```
#####
```

The next section of this report shows the utilization of the various tablespaces. As you may guess from the statistics, this data is from a very small Personal Oracle7 for Windows database. The key columns to keep an eye on here are the `Free bytes` and `%Used`. `Free bytes` shows how much space is available for future extents, but it does not show the size of the free extents that are left. That comes in another section of this report. The `%Used` column is useful because it gives you a feel as to how the amount of free space relates to the amount of total space in the tablespace. Obviously, large tablespaces have a higher likelihood of consuming an extra 10M than a small tablespace does:

```
#####
```

Tablespace Utilization

Table_Space	Free_Bytes	Total_Bytes	%Used
-------------	------------	-------------	-------

```

-----
ROLLBACK_DATA          2,734,080      3,145,728    13.0
SYSTEM                  3,192,832     10,485,760    69.5
TEMPORARY_DATA         2,095,104      2,097,152
USER_DATA               3,065,856      3,145,728     2.5

```

```
#####
```

The next section of this report shows the size of the free segments in the database. Again, the small test instance is relatively empty and suffers from little fragmentation. (In fact, I had to delete one table so that the report would have the 14K free segment that shows up under the user_data tablespace.) If your tablespace typically has extent sizes on the order of megabytes and your free segments are all smaller than this, you probably should consider de-fragmenting the tablespace:

```
#####
```

Tablespace Free Segments Report

```

Table Space              Bytes
-----
ROLLBACK_DATA           2,734,080
SYSTEM                   3,192,832
TEMPORARY_DATA          2,095,104
USER_DATA                14,336
USER_DATA                3,051,520

```

```
#####
```

The next section of this report is useful in databases where there is a relatively small (< 100) tables in the instance and you wish to keep track of how each of these tables is growing. This part of the report becomes very large when you install an application such as Oracle Financials, which has several hundred tables to track. It has multiple purposes. First, it lets you see what is in each of the tablespaces and how much size the tables are consuming. By the way, never let any of your users put personal tables in the system tablespace (as this report shows—I should have known better):

```
#####
```

Table Sizes Report

```

Tablespace Name      Table Name      Owner      Bytes
-----
SYSTEM              ORDERS          JGREENE     8,192
SYSTEM              CUSTOMERS       JGREENE     4,096
SYSTEM              PRODUCT_PROFILE JGREENE     4,096
SYSTEM              USER_PROFILE    JGREENE     4,096
USER_DATA           CUSTOMER        SCOTT       8,192
USER_DATA           BONUS          SCOTT       4,096

```

USER_DATA	DEPT	SCOTT	4,096
USER_DATA	DUMMY	SCOTT	4,096
USER_DATA	EMP	SCOTT	4,096
USER_DATA	ITEM	SCOTT	4,096
USER_DATA	ORD	SCOTT	4,096
USER_DATA	PRICE	SCOTT	4,096
USER_DATA	PRODUCT	SCOTT	4,096
USER_DATA	SALGRADE	SCOTT	4,096
USER_DATA	SCOTTTEST	SCOTT	4,096

#####

The next section provides information about the indexes stored in the database. Once again, it provides a breakdown by tablespace, owner, and size. You definitely should be looking out for objects placed in the wrong tablespace (such as the CUSTOMERS_IDX index put there by that JGREENE fellow—would you believe that I did it intentionally just so that I would have a good example to talk about in this section?):

#####

Index Sizes Report

Tablespace Name	Index Name	Owner	Bytes
SYSTEM	CUSTOMERS_IDX	JGREENE	4,096
USER_DATA	CUSTOMER_PRIMARY_KEY	SCOTT	4,096
USER_DATA	DEPT_PRIMARY_KEY	SCOTT	4,096
USER_DATA	EMP_PRIMARY_KEY	SCOTT	4,096
USER_DATA	ITEM_PRIMARY_KEY	SCOTT	4,096
USER_DATA	ORD_PRIMARY_KEY	SCOTT	4,096
USER_DATA	PRICE_INDEX	SCOTT	4,096
USER_DATA	PRODUCT_PRIMARY_KEY	SCOTT	4,096

#####

The next section of the utilization report shows fragmented (more than one extent) objects in the database. As mentioned earlier in this book, there may be some cases where small amounts of fragmentation are not a problem. However, when tables or indexes start to approach their maximum number of extents (which I like to keep the same for all tables and indexes in the instance so that I know how close I am getting to my limit), it is time to start planning some time for de-fragmentation:

#####

Table/Index Fragmentation Report

Owner	Object Name	Table Index	Bytes Used	FRAGS
JGREENE	ORDERS	TABLE	8,192	2
SCOTT	CUSTOMER	TABLE	8,192	2

#####

The final section of this report has nothing to do with utilization; however, it is handy to have when you are figuring out what to do if you are heading toward problems. It is a listing of the data files in the database. In larger instances, it can be a useful reference as to which tablespaces are stored on which disk drives:

#####

Data File Listing Report

Table Space	File Name	Bytes
ROLLBACK_DATA	C:\ORAWIN\DBS\wdbrrbs.ora	3,145,728
SYSTEM	C:\ORAWIN\DBS\wdbsys.ora	10,485,760
TEMPORARY_DATA	C:\ORAWIN\DBS\wdbtemp.ora	2,097,152
USER_DATA	C:\ORAWIN\DBS\wdbuser.ora	3,145,728

#####

A few final remarks on this report. If you are curious about the actual SQL code used to generate these reports, it can be found on the enclosed disk. You can use any ASCII text editor or reviewer to take a look at it. The actual SQL tends to be rather simple, although most of the work is in setting up the formatting and sizing of the various columns. This work tends to keep things to one line per item and make the data much easier to read. Feel free to copy this report and modify it to suit your personal tastes.

TUNING MONITORING

The next report in this family focuses on the parameters that are most likely to require tuning in your Oracle instance. The guidance for these queries comes from the Oracle Server Administrator Guide, Oracle books on tuning in the UNIX environment and sources such as the CompuServe Oracle bulletin board. It captures the parameters that Oracle and working DBAs have found to cause problems, along with (most importantly) the recommended values for these parameters. (I decided early on to include these limits and the corrective action for out-of-specification readings because it was just too hard to remember which limit was associated with which reading.)

This report starts out with a heading just like the previous report. You know generally speaking what that heading looks like, so let's move on in to the actual report itself. The first section of this report covers the key memory areas that Oracle uses. As you can see, each of these has a measured set of values against a goal. In this case, the data dictionary cache miss ratio is 17 percent, which is above the goal

of 10 percent. It also indicates that you should increase the `shared_pool_size` tuning parameter in your `init.ora` file (again, this report is designed to make it easy to see what tuning actions are needed):

```
#####
```

Memory Allocation Checks

```
#####
```

Library Cache Check

Goal: <1%

Corrective Action: Increase `shared_pool_size`.
Write identical SQL statements

Library Cache Miss Ratio (%)

```
-----  
                                .33
```

```
#####
```

Data Dictionary Cache Check

Goal: <10%

Corrective Actions: Increase `shared_pool_size`

Data Dictionary Cache Miss Ratio (%)

```
-----  
                                17.09
```

```
#####
```

Multi-Threaded Server Session Memory

Goal: `Shared_pool_size` at least equal to maximum.
session memory

Corrective Action: Increase `shared_pool_size`

Session Memory (Bytes)

```
-----  
                                49,152
```

Shared_Pool_Size (Bytes)

```
-----  
                                3,500,000
```

```
#####
```

Buffer Cache Hit Ratio

Goal: Above 60 to 70 percent

Corrective Action: Increase db_block_buffers

Hit Ratio (%)

```
-----
          92.0
```

```
#####
```

The next section looks at disk input/output loading from the Oracle instance's point of view. It also has the blessing of separating read and write operations. If the instance is write-intensive, you should make sure that you have balanced the input/output load with the rollback segments and online redo log files. In the case of the Personal Oracle7 for Windows instance, there is only one disk, so you cannot do much to balance this load. This brings up a good point. If you consistently include the name of the disk drive in your directory names, you will be able to map a given file easily with its read and write operations to a disk drive (for example, you make a UNIX directory /d55 as the mount point for disk 55 instead of /oracle3 and put the data files in directories such as /d55/oradata/prod):

```
#####
```

Disk I/O Checks

```
#####
#####
```

Disk Activity Check

Goal: Balance Load Between Disks

Corrective Action: Transfer files, reduce other loads to disks, striping disks, separating data files and redo logs.

Data File	Reads	Writes
-----	-----	-----
C:\ORAWIN\DBS\wdbsys.ora	268	
C:\ORAWIN\DBS\wdbuser.ora		
C:\ORAWIN\DBS\wdbrrbs.ora	2	
C:\ORAWIN\DBS\wdbtemp.ora		

```
#####
```


The next section of this report discusses contentions that may occur within the Oracle instance due to a lack of resources. As with the other portions of this report, each of these sections provides you with a goal and a corrective action if you exceed your goal. After a while, you will learn to scan for the goal quickly and compare it with the actual values that you have read:

```
#####
```

Oracle Contention Checks

```
#####
#####
```

Rollback Segment Contention

Goal: Measured Counts < 1% of total gets
(the choice of Oracle column names makes it impossible to do this calculation for you)

Corrective Action: Add more rollback segments

```

Total Gets
-----
          5,739
```

```

Class                      Counts
-----
system undo header
system undo block
undo header
undo block
```

```
#####
```

Latch Contention Analysis

Goal: < 1% miss/get for redo allocation
< 1% immediate miss/get for redo copy

Corrective Action: Redo allocation- decrease log_small_entry_max_size
Redo copy- Increase log_simultaneous_copies

Latch Type	Misses/Gets (%)	Immediate Misses/Gets (%)
redo allocation	.00000	.00000
redo copy	.00000	.00000

#####

MTS Dispatcher Contention

Goal: < 50%

Corrective Action: Add dispatcher processes

#####

Shared Server Process Contention

Goal: Shared processes less than MTS_MAX_SERVERS

Corrective Action: Alter MTS_MAX_SERVERS

Shared Server Processes

0

Latch Type MTS_MAX_SERVERS

mts_max_servers 0

#####

Redo Log Buffer Space Contention

Goal: Near 0

Corrective Action: Increase size of redo log buffer

Requests

#####

Sort Memory Contention

Goal: Minimize sorts to disk

Corrective Action: Increase sort-area-size

Type	Number
sorts (memory)	148
sorts (disk)	

#####

Free List Contention

Goal: Number of counts less than 1% of total gets

Corrective Action: Increase free lists (per table)

Total Gets
6,293

Class	Counts
free list	

#####

The final section of this report shows measurements of the loading of the computational capacity and memory from the operating system perspective. Again, the printout shows you a goal versus the actual measurement. You can use this to check for situations wherein the operating system is heavily loaded while Oracle is not or vice versa to determine whether other applications or operating system tuning might be the problem. If you are not using a UNIX operating system, it's a good idea to show the monitoring scripts to your friendly system administrator, who should be able to come up with something equivalent in your world.

One final point to keep in mind when running these reports. The goals that are set here are not your opinion or my opinion. These values have been gathered from Oracle vendor documents. Therefore, if people do not like your results, they are not arguing with you or me. They are arguing with the guys who wrote the RDBMS itself. That kind of credibility can be useful in solving what sometimes turns into endless discussions as to what your numbers really mean, especially when people are sensitive to conclusions such as the system needs more memory.

SECURITY MONITORING

The third in the monitoring series of reports deals with security. This is not a report that has a predefined set of goals and corrective actions. You have to compare the actual security roles and privileges granted against your own security plan. No

outsider can know which privileges are correct for your accounting supervisions. However, reading this report is not as hard as it may seem. You get used to the privilege grants that should be given to an individual role. Items such as an end user being granted DBA privileges tend to jump out at you.

The first section of this report shows the restrictions placed on user accounts via the profile mechanism. I have not used this mechanism frequently, but put it in the report for those of you who might use it:

#####

Profiles

PROFILE	RESOURCE_NAME	LIMIT
-----	-----	-----
DEFAULT	COMPOSITE_LIMIT	UNLIMITED
DEFAULT	CONNECT_TIME	UNLIMITED
DEFAULT	CPU_PER_CALL	UNLIMITED
DEFAULT	CPU_PER_SESSION	UNLIMITED
DEFAULT	IDLE_TIME	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	UNLIMITED
DEFAULT	PRIVATE_SGA	UNLIMITED
DEFAULT	SESSIONS_PER_USER	UNLIMITED
DEFAULT	COMPOSITE_LIMIT	UNLIMITED
DEFAULT	CONNECT_TIME	UNLIMITED
DEFAULT	CPU_PER_CALL	UNLIMITED
DEFAULT	CPU_PER_SESSION	UNLIMITED
DEFAULT	IDLE_TIME	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	UNLIMITED
DEFAULT	PRIVATE_SGA	UNLIMITED
DEFAULT	SESSIONS_PER_USER	2

#####

The next section of the report shows access to the very powerful system privileges in your instance. If you use roles, you should grant your system privileges to this relatively small number of roles. When reviewing this list, you need to be careful about the create and delete privileges that should be restricted to developers and the “any” privileges that are normally restricted to DBAs. This section can be fairly long, so only a small portion of it is presented here as an example:

#####

System Privileges

GRANTEE	PRIVILEGE	ADM
-----	-----	---
DBA	RESTRICTED SESSION	YES
DBA	SELECT ANY SEQUENCE	YES

DBA	SELECT ANY TABLE	YES
DBA	UPDATE ANY TABLE	YES
GOLFER	CREATE SESSION	NO
GOLF_PRO	CREATE SESSION	NO

#####

The next two sections of the report list your users and roles. The key in the user section is to ensure that the default and temporary tablespaces are set up so that users are creating objects where you want. In the roles section, you are usually just making sure that all the roles that you expect are there:

#####

Users

USERNAME	DEFAULT_TABLESPACE	TEMPORARY_TABLESPACE
-----	-----	-----
BSMITH	USER_DATA	TEMPORARY_DATA
JGREENE	USER_DATA	TEMPORARY_DATA
JSMITH	USER_DATA	TEMPORARY_DATA
SCOTT	USER_DATA	TEMPORARY_DATA
SYS	SYSTEM	SYSTEM
SYSTEM	SYSTEM	SYSTEM

#####

Roles

ROLE	PASSWORD
-----	-----
CONNECT	NO
DBA	NO
EXP_FULL_DATABASE	NO
GOLFER	NO
GOLF_PRO	NO
IMP_FULL_DATABASE	NO
RESOURCE	NO

#####

The next section is important for databases that use roles to grant privileges. It shows which roles are granted to the individual users. There are two other key columns. The first one, ADM, shows whether the user is allowed to grant that role to other users. This could be a security hole if granted to the wrong individuals. The next column, DEF, shows whether this is a default role for the user or whether the user has to issue a command to set up his session to use that role:

#####

Role Privileges

GRANTEE	GRANTED_ROLE	ADM	DEF
DBA	EXP_FULL_DATABASE	NO	YES
DBA	IMP_FULL_DATABASE	NO	YES
JGREENE	DBA	NO	YES
JGREENE	GOLFER	YES	YES
JGREENE	GOLF_PRO	YES	YES
JGREENE	RESOURCE	NO	YES
SCOTT	CONNECT	NO	YES
SCOTT	RESOURCE	YES	YES
SYS	CONNECT	YES	YES
SYS	DBA	YES	YES
SYS	EXP_FULL_DATABASE	YES	YES
SYS	IMP_FULL_DATABASE	YES	YES
SYS	RESOURCE	YES	YES
SYSTEM	DBA	YES	YES

#####

The next section of this report can go on for several pages and is perhaps the hardest section to proofread. It lists the tables in the instance and what object access privileges have been granted on them. The key here is to make sure that these grants match your security scheme. This could be difficult for a complex scheme; however, in practice you check for things such as the following: only supervisors have write privileges to reference tables, no one has delete privileges on any tables, and so forth. This takes some time, but is the only way to guarantee that your security scheme has been properly implemented:

#####

Table Privileges

GRANTEE	TABLE_NAME	OWNER	PRIVILEGE
JGREENE	DBMS_ALERT	SYS	EXECUTE
JSMITH	GOLF_SCORES	JGREENE	SELECT
JSMITH	GOLF_SCORES	JGREENE	UPDATE
JSMITH	GOLF_SCORES	JGREENE	INSERT
BSMITH	GOLF_SCORES	JGREENE	SELECT
BSMITH	GOLF_SCORES	JGREENE	UPDATE
BSMITH	GOLF_SCORES	JGREENE	INSERT
PUBLIC	ACCESSIBLE_TABLES	SYS	SELECT
PUBLIC	ALL_CATALOG	SYS	SELECT
PUBLIC	ALL_COL_COMMENTS	SYS	SELECT
PUBLIC	ALL_COL_GRANTS_MADE	SYS	SELECT

#####

The final four sections of this report list the views, packages, indexes, and synonyms used in the database. These can be useful references when you are reading some of the other sections, so they are included here. They are typically not directly involved with the security scheme.

This is the basic security report. The SQL that executes these queries is stored as `secchk.sql` on the enclosed disk. As with the other reports, feel free to make a copy of this report and modify it to fit your individual needs. Perhaps you have a specific problem that you wish to check for quickly and put it at the beginning of the report.

CONFIGURATION MONITORING

The final monitoring report in this series scans the Oracle internal configuration tables and reports the values of the parameters as they actually are in the working database. This can be very useful because many of your working parameters are not explicitly listed in your `init.ora` and `config.ora` files. You therefore inherit the default values for your operating system. This report shows you the operational value of the parameter—whether it is derived from the operating system default or an explicit listing in the initialization files.

The first section of the report lists the data files associated with the instance. Because data storage is the function of a database, the configuration of these files is understandably important. This can be useful when comparing configurations before and after problems arise. For example, if everything is working fine until after you add your 129th data file (with 128 being a nice even multiple of 2), you might want to mention this to Oracle support (you may have exceeded an internal limit or found a new bug):

```
#####
```

Data Files Associated With This Database

ID	File Name	Tablespace	Bytes
3	C:\ORAWIN\DBS\wdbrrs.ora	ROLLBACK_DATA	3,145,728
4	C:\ORAWIN\DBS\wdbtemp.ora	TEMPORARY_DATA	2,097,152
2	C:\ORAWIN\DBS\wdbuser.ora	USER_DATA	3,145,728
1	C:\ORAWIN\DBS\wdbsys.ora	SYSTEM	10,485,760

```
#####
```

The next section lists the redo log files in your instance. This is more than just a listing of filenames. It contains the group number, which shows how these files relate to one another when you are using mirrored redo log files. The status column can also be important if there is a problem. If you notice that there is an error message or offline indicator next to one of these log files, you need to investigate it:

#####

Redo Log Files

GROUP#	STATUS	MEMBER
2		C:\ORAWIN\DBS\wdblog2.ora
1		C:\ORAWIN\DBS\wdblog1.ora

#####

The next section shows the tablespaces in the instance. However, rather than looking at how full they are or what objects they contain, this report focuses on their default parameters and status. Remember, if you do not specify the storage parameters when you create a table or index, you inherit these defaults automatically:

#####

Tablespace Information

Tablespace	Init/Next	Min/Max	PCT	STATUS
SYSTEM	4096/4096	1/121	10	ONLINE
USER_DATA	4096/4096	1/121	10	ONLINE
ROLLBACK_DATA	102400/102400	2/121	50	ONLINE
TEMPORARY_DATA	4096/4096	1/121	10	ONLINE

#####

The next two sections are designed to provide a count of database objects by type and also list some of the nondata objects. I put these queries in this report to capture some of the objects that you do not normally see, such as rollback segments. You may have accepted default values when you created the instance and never thought of them again until a problem arises. The data on this report can help you reconstruct objects and figure out what is wrong with one of these relatively obscure objects:

#####

Non-Data Objects Listing

Object name	Obj Type	Bytes Used	SUM(BLOCKS)	#
1.377	CACHE	2,048	1	1
RB1	ROLLBACK	204,800	100	2
RB2	ROLLBACK	204,800	100	2
RB_TEMP	ROLLBACK	542,720	265	53
SYSTEM	ROLLBACK	204,800	100	4

#####

Numbers of Database Objects by Type

Object Type	Count
-----	-----
??	56
CLUSTER	8
COLUMN	3,805
CONSTRAINT	347
DB LINKS	
SEQUENCE	12
SYNONYM	366
TABLE	96
VIEW	414

sum	5,104

#####

The next section shows the Oracle background processes that are running. This is an important consideration when you are trying to improve performance through the use of parallel processing. For example, as you raise the number of multi-threaded servers (additional processes running in parallel to service user requests) you may start to encounter problems. This report documents your configuration as you increase the number of processes so that you can know what works best on your system:

#####

Oracle Background Processes Running

PADDR	BGProc	DESCRIPTION	ERROR
-----	-----	-----	-----
814CE4E8	PMON	process cleanup	
814CE690	DBWR	db writer process	
814CE838	LGWR	Redo etc.	
814CE9E0	SMON	System Monitor Process	

#####

The final two sections of the report capture the configuration of that memory area that you cannot see or touch, but that Oracle depends on for almost all database processing functions—the SGA. Only a small portion of the second query is presented. It goes on for some time listing a number of parameters that you never even knew that you had. Much like the other sections of this report, it captures values that control how your system functions; however, you may not be sure where they are stored and how to access them:

```
#####
```

SGA Sizing

SGA Group Name	Value - Bytes
-----	-----
Fixed Size	36,432
Variable Size	3,846,292
Database Buffers	819,200
Redo Buffers	65,596
-----	-----
sum	4,767,520

```
#####
```

SGA Parameter Listing

Num	Parameter Name	Parameter Value
-----	-----	-----
6	processes	50
7	sessions	60
8	timed_statistics	FALSE
9	resource_limit	FALSE
10	license_max_sessions	25
11	license_sessions_warning	0
12	single_process	FALSE
14	event	
15	shared_pool_size	3500000
16	pre_page_sga	FALSE
20	enqueue_resources	155
32	nls_language	AMERICAN
33	nls_territory	AMERICA
34	nls_sort	
35	nls_date_language	
36	nls_date_format	

```
#####
```

THIRD-PARTY TOOLS

Some of you may feel uncomfortable with the somewhat home-grown nature of these reports. Scripts such as this are used by DBAs all around the world because they were the only way to implement a monitoring program. Oracle itself did not provide a convenient tool to get this information in a presentable format. However, this need has not gone unnoticed in the marketplace. There are a number of vendors who now offer products that are designed to perform this and even more detailed monitoring. There are some with fancy graphics and other features.

These utilities hold promise in certain installations; however, you have to look at a number of them to determine which suits your taste. For example, some of these tools go on for page after page showing all the parameters in the Oracle database. This is thorough, but it is very difficult to sort out the parameters that are important from the others. You also may have budget limitations that need to be considered or want to integrate Oracle monitoring with operating system and network monitoring. The scripts presented in this section have the blessing of depending only on SQL*Plus, which is shipped with every Oracle installation. Even if you want a more sophisticated tool, you can use these scripts while you are evaluating what is on the market.

SUMMARY

This chapter has taken on the ambitious task of describing the concepts behind setting up a program to monitor your Oracle instance provocatively. I got on my soap box about being proactive; however, I really think that your life is easier when you are not reacting in crisis mode. This chapter then presented a series of scripts, with tips as to what to look for and why each section is important. Take some time to study the sample outputs, even though they were done on a lightly loaded Personal Oracle7 for Windows instance on a PC. Once you get comfortable with what is being presented and what you need to look for, you can review these reports quickly to get the answers you need.



- Overview
- Oracle Auditing Events
- Auditing and Performance
- Auditing as Part of Security Monitoring
- Deciding What to Audit

CHAPTER 31

Auditing



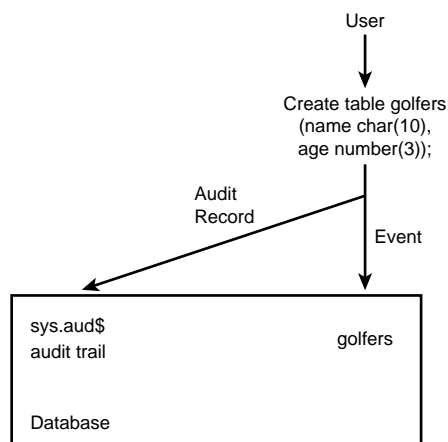
Monitoring provides you with a picture of the status of the database—its utilization, tuning, security grants, and configuration. Auditing is a complementary security tool that helps you record specific activities that the users perform. Obviously, in a busy database, you do not have the time or recording space to keep track of everything. This chapter explains how to set up auditing, what your auditing options are, and how to devise a plan for what you are going to audit.

I have worked on only one database instance where auditing was used. For the most part, I separated the development and production instances. The production instances had only a handful of trusted users who could write to the database. The instance was monitored through the security report, and auditing was not required. In transaction processing instances and those where you have a wide mix of users, auditing can be a useful tool.

OVERVIEW

The key to understanding Oracle auditing is understanding the concept of events. Most of the documentation on auditing starts with discussions of options on where you store the audit trail and what the options are on the audit commands. Let's look at an overview of the concept of events, because the audit trail is a record of specific events. In simplest terms, events are actions taken by the users. These events match up well with system and object privileges within Oracle. These privileges are permissions to perform specific activities or events. For example, `CREATE TABLE` is a privilege that you may grant to your users. If you tell Oracle that you wish to audit `CREATE TABLE`, Oracle records every time any user executes the `create table` command. This is an event (see Figure 31.1).

Figure 31.1.
Oracle auditing and events.



Although being able to track events linked to your system privileges is useful, it may not be enough. For example, suppose you have a large instance with many tables. Most of those tables are routinely updated by average users and there is a high volume of traffic. Perhaps you are interested in tracking access to a few key security tables, but not the many thousands of transactions to the routine tables. Oracle auditing can accommodate your needs by enabling you to audit a given statement (for example, `select`) applied to an individual database table or view. This becomes important for two reasons. First, it takes storage space to keep the records of the audited events, and in most instances, storage space is not unlimited. Second, you do not want to go through an enormous number of records to extract the data that you want. Therefore, Oracle's capability of auditing events on particular database objects can be very useful.

So far, you have explored Oracle's capability of auditing general classes of events (for example, all `create table` commands) and events related to specific objects (for example, all `select` statements on a particular table). The final class of auditing that you can perform enables you to specify that you want a group of events audited with a single statement. For example, suppose that you want to record all user account maintenance activity. Oracle provides an audit option called `user`, which captures all events wherein someone issues a `create user` command, an `alter user` command, or a `drop user` command. You could achieve the same goal by issuing the three individual privilege commands, but the `user` option provides you with a shortcut. There is also a higher level that takes larger groups of these events (`connect`, `resource`, `dba`, and `all`). This can be somewhat confusing because they chose the same names as are used for the default roles provided with Oracle. However, the privileges that are audited are similar to the default privileges granted to these roles. You just have to remember that if you alter your privilege scheme, the audit options in these groups do not change. The next section goes into more detail on the audit options and groups of audit options.

The next topic is setting up Oracle auditing. By default, you do not have auditing turned on in your Oracle instance. This makes good sense. Most new DBAs have trouble dealing with the many log files on the system that are not emphasized in the basic literature. These logs continue to grow until either a problem occurs (and tech support asks you to read them) or you run low on disk space and go looking for culprits. You could easily fill up your tablespaces with audit information if Oracle came with auditing turned on by default. Therefore, you have to tell Oracle that you want to turn auditing on and specify exactly what you want to audit.

There are two steps that you need to take to turn on Oracle auditing. First, you enable the auditing features by going into your initialization file and setting the `AUDIT_TRAIL` parameter. There are three values for this parameter—`DB`, `OS`, and `NONE`. The `NONE` value is pretty obvious and corresponds to auditing being turned off. The `OS` value for the `AUDIT_TRAIL` parameter specifies that you wish to record auditing

information to an operating system file. The `DB` value indicates that you wish to store the audit records in the `SYS.AUD$` table in your database.

This brings up another interesting discussion. Oracle enables you to write auditing records to operating system files on certain operating systems. This is usually designed to use the same file as that used by the operating system auditing utilities. This gives you a single place where all of the auditing records are stored. Usually, these files are under the control of a separate security administrator who is performing an independent review of the actions of all the users and its administrators. If you do not have this kind of external security program, you probably will want to use the internal Oracle tables for auditing. They can be secured so that only the right people get access to them and they have the advantage of being accessible through SQL queries under SQL*Plus or report generation packages. This is much better than having to read through thousands of lines of text in a flat file on the operating system.

So now you have edited your initialization files to set the `AUDIT_TRAIL` parameter to the desired value. You, of course, have to shut down and restart the instance in order to get the new initialization parameter to take effect. However, recall that there are two steps to the process. The second step involves setting up a number of views within the Oracle database that make it easy for you to get access to the information that you desire. It is difficult to issue queries to the internal Oracle data dictionary tables to find out about objects, but it is more difficult to figure the details of the SQL statement to find out about a particular type of auditing. Therefore, Oracle has established a series of easy-to-read views that target specific types of auditing and stores these views in the `CATAUDIT.SQL` directory under the `rdbms` administrative subdirectory under `ORACLE_HOME`. You should run this script as `SYS` to set these views up because they do make dealing with auditing much easier. This script sets up the following audit views for the `DBA`:

- ♦ `AUDIT_ACTIONS`
- ♦ `ALL_DEF_AUDIT_OPTS`
- ♦ `DBA_STMT_AUDIT_OPTS`
- ♦ `DBA_OBJ_AUDIT_OPTS`
- ♦ `DBA_AUDIT_TRAIL`
- ♦ `DBA_AUDIT_SESSION`
- ♦ `DBA_AUDIT_STATEMENT`
- ♦ `DBA_AUDIT_OBJECT`
- ♦ `DBA_AUDIT_EXISTS`

One further note about setting up the audit trail: You need to consider the security of the audit trail itself when you are setting up your system. Imagine the situation

where someone has access to a DBA privileged account and wants to do some bad things. If you do not take steps to prevent it, this person can access Oracle, do all those bad things, and then merely delete the audit records from SYS.AUD\$ that show the bad things (because the person has DELETE ANY TABLE as a DBA). Therefore, it is best to grant DELETE ANY TABLE only to those responsible for database security monitoring (the security administrator or DBA). The same rule goes for operating system files that are used for audit records.

Finally, before you learn the various auditing options that are available to you, it might be useful to explore the information that you will get from the auditing process. Because there is such a wide range of auditing options, the exact data captured varies with the type of auditing being performed. However, there are a few bits of data that are common to all auditing:

- ◆ The Oracle user ID that was used to execute the statement/event that was being audited
- ◆ The date and time that the event occurred
- ◆ A number (which you would have to translate unless you use the views) showing the statement that was executed
- ◆ The database objects that were referenced in the statement that was executed

This concludes the overview discussion on auditing. By now, you should be comfortable with what auditing can provide for you in general terms. You should be able to set up auditing within Oracle, although the exact statements to audit for particular events will have to wait until the next section. Finally, you should know the views that Oracle provides to review audit data and be able to set these views up in your Oracle database. In the next section, you learn the statements and options that you can use to start auditing given events.

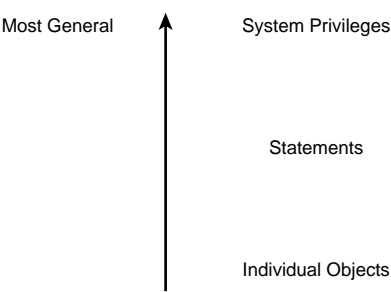
ORACLE AUDITING EVENTS

As mentioned earlier, Oracle enables you to look at three different levels of detail when auditing (see Figure 31.2). At the highest level, it tracks when you use one of your system privileges (for example, ALTER SYSTEM). At the next level, it enables you to track the execution of a given type of SQL statement (for example, select). Finally, it enables you to look at specific SQL commands applied to specific database objects (for example, delete on SYS.AUD\$ table). Oracle also provides some group audit options that cover sets of commands that are commonly audited at the same time. These serve as shortcuts for you because you could issue commands to audit each of these events individually. In this section, you look at each of these types of auditing to provide you with a picture of what you can do with Oracle's auditing services.

The first level of auditing is the very high level related to the use of Oracle system privileges by a user. This is relatively straightforward because you have to know only what the Oracle system privilege is that you want to audit and then issue a command similar to the following:

```
audit update any table;
```

Figure 31.2.
Oracle auditing
hierarchy.



The second level of auditing captures information about individual statements that the users are issuing. Table 31.1 lists the statement auditing options that are available to you.

TABLE 31.1. ORACLE STATEMENT AUDITING OPTIONS.

<i>Audit Option</i>	<i>Statements Captured in the Audit</i>
alter sequence	alter sequence
alter system	alter system
alter table	alter table
cluster	create cluster
	alter cluster
	truncate cluster
	drop cluster
comment table	comment on [table]
database link	create database link
	drop database link
delete table	delete table
execute procedure	EXECUTION OF FUNCTIONS AND PROCEDURES
grant procedure	GRANT OR REVOKE ON A PROCEDURE
grant sequence	GRANT OR REVOKE ON A SEQUENCE

<i>Audit Option</i>	<i>Statements Captured in the Audit</i>
grant table	GRANT OR REVOKE ON A TABLE
index	create index alter index drop index
insert table	insert into
lock table	lock table
not exists	ALL STATEMENTS RETURNING NON-EXISTENCE ERRORS
procedure	create function create package create package body create procedure drop function drop package drop procedure
public database link	create public database link drop public database link
public synonym	create public synonym drop public synonym
role	create role alter role set role drop role
rollback segment	create rollback segment alter rollback segment drop rollback segment
select sequence	SELECT STATEMENT ON A SEQUENCE
select table	SELECT STATEMENT ON A TABLE
sequence	create sequence drop sequence
session	ALL CONNECTIONS AND DISCONNECTIONS
synonym	create synonym drop synonym

continues

TABLE 31.1. CONTINUED

<i>Audit Option</i>	<i>Statements Captured in the Audit</i>
system audit	audit
	no audit
system grant	ALL SYSTEM PRIVILEGE AND ROLE GRANTS
	ALL SYSTEM PRIVILEGE AND ROLE REVOKES
table	create table
	truncate table
	drop table
tablespace	create tablespace
	alter tablespace
	drop tablespace
trigger	create trigger
	alter trigger with enable or disable
	alter table with enable, disable or drop
update table	UPDATE STATEMENT ON A TABLE
user	create user
	alter user
	drop user
view	create view
	drop view

You can activate statement level auditing with a command line that contains the audit option from Table 31.1, similar to the following:

```
audit role;
```

The final auditing layer is designed to give you the most precise control over what is audited: capturing information about individual objects. It is ideal if there are only a few tables that are like security problems. There is nothing to prevent you from combining auditing of a few tables with more general auditing for the use of special privileges that are normally restricted to DBAs. Table 31.2 shows the auditing at the individual object level that Oracle permits you to perform.

TABLE 31.2. ORACLE OBJECT AUDITING OPTIONS.

<i>Object</i>	<i>Auditing Permitted</i>
procedure	audit
	execute
	grant
	rename
sequence	alter
	audit
	grant
snapshot table	select
	select
	alter
	audit
	comment
	delete
	grant
	index
	insert
	lock
	rename
	select
	update
	audit
	comment
	delete
view	grant
	insert
	lock
	rename
	select
	update

To activate object auditing, you need to give the `audit` command, the object auditing option, and the object that is being audited similar to the following example:

```
audit delete on golf_scores;
```

In addition to these three auditing options, Oracle provides four special auditing options that are actually groups of some of the statement privileges. Three of these have the same name as that of the default roles provided in Oracle. They are different and are not changed when you change the privilege sets assigned to these default roles. These audit option groups are listed in Table 31.3.

TABLE 31.3. AUDIT GROUPS (SHORTCUTS).

<i>Audit Group Name</i>	<i>Encompassed Statement Audit Options</i>
all	alter system
	cluster
	database link
	index
	not exists
	procedure
	public database link
	public synonym
	role
	rollback segment
	sequence
	session
	synonym
	system audit
	system grant
	table
	tablespace
	trigger
	user
	view
dba	system audit
	public database link
	public synonym
	role

<i>Audit Group Name</i>	<i>Encompassed Statement Audit Options</i>
	system grant
	user
connect	session
resource	alter system
	cluster
	database link
	procedure
	rollback segment
	sequence
	synonym
	table
	tablespace
	view

So far, there have been a lot of different auditing options presented. Although this list has been long, it has not been very selective. It will capture every time someone selects against one of the tables in the instance or every time someone tries to delete from a particular view. In a busy instance, this can still produce a large audit trail, and it would take some of your valuable time to sift through it. Therefore, Oracle provides two optional clauses (*whenever* and *by*), that help you limit the scope of your auditing even further. The *whenever* clause has two options—*whenever successful* and *whenever not successful*. When you use one of these clauses, audit records are met only when the statement completes successfully or fails. The *by* optional clause enables you tell Oracle to write a record only for individual users, to write one record every time the statement is issued, or to write a single record to capture all the times during a session that the user executes the statement(s) being audited. Perhaps some examples are in order:

```
audit session by jsmith;

audit delete on golf_scores by access;

audit delete any table whenever not successful;

audit select any table by session;
```

This discussion is already getting long because auditing is a complex subject with a wide range of options that may come in handy under the right circumstances. There are only two additional topics that relate to what you can audit. The first is the other half of controlling what you can audit. So far, you have learned how you

can add auditing checks into the system. It is necessary to be able to delete auditing checks. This is accomplished with the `noaudit` command, which is the exact opposite and takes options similar to the `audit` command (except for the `by session` and `by access` options). For example, the following `noaudit` commands can be useful:

```
noaudit session by jsmith;
```

```
noaudit delete on golf scores;
```

```
noaudit delete any table whenever not successful;
```

```
noaudit select any table;
```

The last part of this section offers sample outputs of the auditing views. There are a number of these views as outlined in the previous section. If you are unsure from the title as to what the view is doing, look at the fields in the view (using the `desc` command under SQL*Plus) or actually issue the query to see what it looks like. The following outputs show the general concepts. The first one shows which statement audit options I have activated in my database. The other one shows the actual audit trail for sessions in this database (note that I purged the audit trail just before running this command to keep from filling up many pages with this example):

```
SQL> select * from dba stmt audit opts;
```

USER_NAME	AUDIT_OPTION	SUCCESS	FAILURE
JSMITH	CREATE SESSION	BY ACCESS	BY ACCESS
	SELECT TABLE	BY SESSION	BY SESSION
	DELETE TABLE	BY SESSION	BY SESSION
	CREATE CLUSTER	BY ACCESS	BY ACCESS
	CLUSTER	BY ACCESS	BY ACCESS
	TABLE	BY ACCESS	BY ACCESS
	ROLE	BY ACCESS	BY ACCESS
	SELECT TABLE	BY SESSION	BY SESSION
	SELECT ANY TABLE	BY SESSION	BY SESSION

9 rows selected.

```
SQL> select * from dba_audit session;
```

```

OS_USERNAME
-----
USERNAME
-----
USERHOST
-----
TERMINAL
-----
TIMESTAMP ACTION_NAME LOGOFF_TI LOGOFF_LREAD LOGOFF_PREAD
LOGOFF_LWRITE
-----
LOGOFF_DLOCK SESSIONID RETURNCODE
-----
SESSION LABEL

```

OraUser
JGREENE

Windows
04-JUN-95 LOGON

82

0

AUDITING AND PERFORMANCE

Auditing and performance is another subject in which there is no easy answer. The impact on performance of using the auditing command depends on the following factors:

- ◆ The overall load on your system
- ◆ Whether your application issues a large number of statements or a smaller number of very demanding statements
- ◆ The number of events that you audit and how frequently they come up in routine processing

There are probably a few more that you could think up, but you get the general idea. It is another case wherein the impact on performance depends on your particular database server, how you have your database set up, and how your applications are designed. The performance impact is usually negligible if you merely audit events such as connections, disconnects, and unusual activities, (for example, deleting from a key table) that might indicate people are doing something that they should not be doing. If you need more precise auditing, you can think of it as another (relatively small) database update that will occur every time your application issues an audited statement.

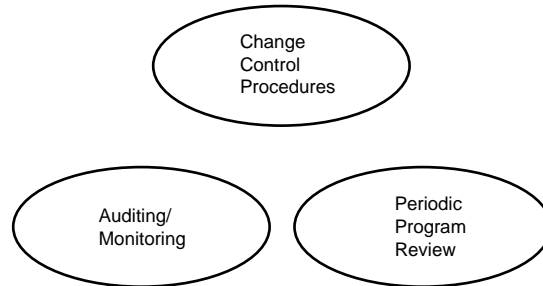
AUDITING AS PART OF SECURITY MONITORING

Auditing has been used for the purpose of recording how much a system is being used by the user community. This task uses auditing to justify the system's costs and benefits to upper management. However, the auditing system is basically designed to capture activity that might indicate a security problem. Therefore, your use of auditing needs to match up with your local security scheme.

An overview of the parts of a security scheme will be useful here (see Figure 31.3). Your location may have some specific guidance that is slightly different than this, but a security scheme usually starts with a documented plan. The plan puts a controlled procedure in place for changes in the computer environment. This procedure may have approvals by supervisors, or it may have a central security administrator controlling all activity. Security schemes usually have some sort of

routine auditing/monitoring feature to ensure that things are going according to plan. Finally, security schemes need to be reviewed periodically in some form of audit to ensure that the program itself is functioning as needed.

Figure 31.3.
Components in a
security scheme.



Your goal when designing an audit program is to ensure that it is just right for your needs. Too much auditing will cost you time to review and slow performance. Too little auditing may miss unauthorized activity—you should be aware of that. Because there is no way to anticipate every possible situation, make your best guess at what to audit and then monitor your program to see whether it is meeting your needs.

DECIDING WHAT TO AUDIT

So far, you have learned the pieces of an auditing program. Now is the time to put these pieces together into a program that will work for you. Auditing can turn into a big deal for some people, but it's a good idea to keep things simple. Therefore, build an auditing program that contains the following pieces:

- ♦ The events being audited
- ♦ Monitoring of the audit trail data
- ♦ Routine purging of audit trail data
- ♦ Action plans when suspicious activity is detected

What should you be auditing? The common recommendation that Oracle puts forward in its literature—which I agree with—is to start auditing generally and then get specific. This enables you to keep an eye on everything in a broad sense. When you detect suspicious activity, you can then tighten your search in that area. Examples of generally suspicious events include the following:

- ♦ Use of the “any” privileges by common users
- ♦ Activities covered under the dba auditing option

- ◆ Creating and dropping tables
- ◆ Anything other than select against the audit trail data file
- ◆ Any alter tablespace commands

When you set up this broad auditing scheme, you will get a number of records in your audit trail. This is natural and normal. Your job is to sort out those activities that you were performing for valid reasons (for example, you altered a tablespace because it ran out of space) from those that are indicative of someone tampering with your database. If you have a lot of activity and therefore have trouble remembering when you log on and perform these activities, you may want to keep a sheet of paper at your desk on which you can quickly record when you perform this type of activity so that you can check it against the audit trail.

This brings up monitoring the audit trail. There are two key issues associated with this subject—how and when. My recommendation for the “how” is to set up a simple SQL script report, similar to those discussed in Chapter 24. I like to develop a shell script and the SQL script for this purpose. The shell script calls the SQL script and can be placed in an automatic job scheduling utility (such as `cron`) to run automatically and then mail me the results. Depending on your host operating system, you may find one of the utilities on the enclosed disk useful for this purpose. The SQL script contains SQL queries against the views discussed earlier in this section that capture the activities that you are monitoring (for example, `DBA_AUDIT_STATEMENT`).

The “when” part of monitoring is somewhat more site-specific. If you are in a low-security environment, a monthly review of the audit trail may be good enough. In other environments, you may want to look at the record on a daily basis. I tend to favor running the monitoring reports on a more frequent basis because you are better able to react to problems and are also better able to remember the legal activities that will be showing up on the auditing reports. (Imagine having to remember all of your tablespace maintenance sessions for a month.)

An activity related to monitoring the audit trail is purging it. Whether you use the system files or the internal Oracle audit table, you need to realize that space is limited. Therefore, a routine program to remove data that is old should be put in place. You have to decide how much data to keep. If you print out your reports or save the report files on disk, you can purge the audit trail in the database or system file any time you want. You may wish to keep the audit trail a little longer if you are using it to produce statistical reports. The SQL to delete older audit records from the Oracle internal auditing table can be something similar to the following:

```
SQL> delete from sys.aud$ where timestamp < to_date('01-jun-95');
```

```
18 rows deleted.
```

The final thing that you need to be prepared for is a plan of action if you detect something suspicious. Perhaps you detect some questionable select activity (for example, you were monitoring `select whenever not successful`). From this, you find a particular user who seems to be issuing queries around your system—perhaps fishing for data. You may choose to audit everything that this user does. This builds a case against the user. Another option is to perform more detailed auditing of all activity related to your most sensitive tables (for example, personnel and monetary records). The key is to figure out what information you want and then add auditing to capture this data. This can be a balancing act to prevent an impact on performance or making your audit reports too long.

SUMMARY

This chapter presented the basics of Oracle auditing, starting with an overview and then progressing into detail about the events that Oracle is able to audit. Next, you learned when and what to audit when you are designing your audit program. This program needs to mesh with your security scheme and minimize any impacts on the overall performance of your system. Finally, you explored some of the considerations of what to do if you detect potentially improper activities.

As I mentioned earlier, I have rarely used Oracle auditing. In most of the instances that I have worked with, the basic permission scheme restricted the capabilities of the users in the production instances to ensure that they could not hurt the instance or access information that they were not entitled to access. However, you may want to consider auditing in instances wherein users have access to flexible query tools (such as SQL*Plus) or you have sensitive information mixed with nonsensitive information.

-
- A detailed black and white illustration of a celestial globe. The globe is tilted, showing the zodiac signs Gemini, Cancer, and Leo. Gemini is depicted as two figures, Cancer as a scorpion, and Leo as a lion. A large, bold 'MAR' (March) is written across the lower left. A band labeled 'TEMPERATE' runs diagonally across the middle. The globe is surrounded by a circular border with degree markings. In the background, there are illustrations of a city and a landscape.

- What Is Tuning?
- What Can You Control?
- Host Computer Indicators
- Oracle Resource Contention
- A Tuning Checklist

CHAPTER 32

[illegible]

Now for a really fun chapter—tuning your database. True, if you set your initialization parameters incorrectly, you cannot even start up your instance, but this is the type of stuff that separates the true DBAs who are masters of their databases from the caretakers who do whatever their databases want them to do. Tuning is a powerful capability that enables you to get the most out of your existing hardware and software. Once you master tuning, you can stand up in meetings and explain why any observed poor performance is not a problem with the database.

The concept of tuning the Oracle instance comes from one of the basic requirements designed into Oracle—flexibility. Tuning in flat file systems usually consisted of making the applications as efficient as possible. Tuning in early PC databases such as dBase was probably beyond the concept of most of the people who wanted an easy-to-use tool. However, when databases such as Oracle came along, tuning became a necessity. Oracle has to work in a wide variety of environments while supporting a wide variety of applications. It also has to work well in a multi-user environment that scales up to some very large environments (hundreds of users and hundreds of gigabytes of data). Obviously, settings and algorithms that work well for small transaction databases are not the best for large data warehouses. Therefore, the Oracle software actually consists of a number of different algorithms that are selected as part of the tuning process.

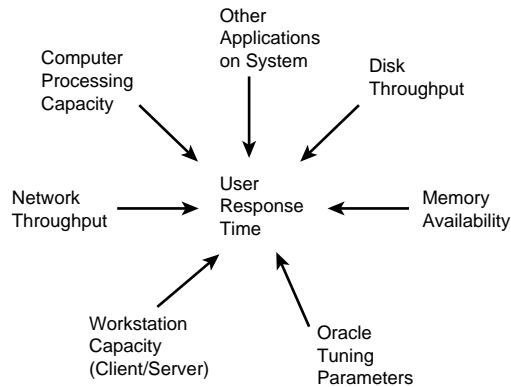
This chapter introduces the basic concepts of tuning an Oracle database. It should be good enough for the vast majority of DBAs. There are entire books devoted to the subject of tuning, and they may come in handy for those working in especially demanding areas such as data warehouses or multimedia. This chapter starts with a discussion of what tuning is, then lists the major things that you can control when working with your Oracle database. You explore the host computer indicators that may be useful in determining whether you have an Oracle tuning problem or not. Next, you learn about Oracle resource contention and how to tune it out. Finally, you look at a checklist that may be handy when tuning your Oracle instance. This is an ambitious chapter, so let's jump right into it.

WHAT IS TUNING?

Recall that Oracle has different internal algorithms that it can use and that you can somehow control these to make things run more smoothly. Recall also that there are settings in the configuration files that you can adjust. Tuning is all of these things and more. It is taking the whole Oracle system and seeing what can be done to improve performance. There may be times when the answer to improving performance lies with the system administrator (adding additional disk drives) or developers (making the software use more efficient algorithms).

There are issues that have relatively simple answers. For example, when someone asks you whether you can interface Visual Basic on a PC to your Oracle database, you can answer that you need to install SQL*Net on both machines and set up the ODBC drivers on the PCs. If everything were that simple, life would be wonderful. However, if you have users who think that the response time of the system is not adequate for their particular reports, it is a much more complex issue (see Figure 32.1). In this situation, there are a number of possible answers as to why the system performs as it does. In the vast majority of cases, people will first point the finger at the database being too slow.

Figure 32.1.
Factors that contribute to system performance.



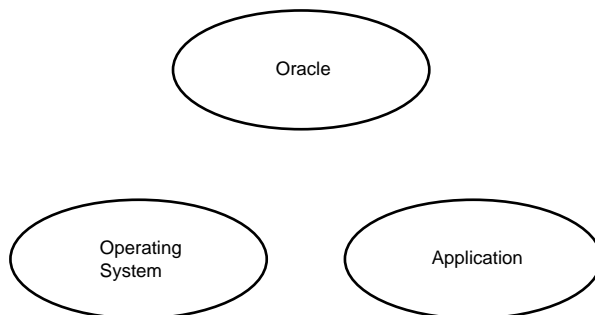
This brings up the first important part of a tuning exercise—determining how well the system is performing. Before automobiles became complex computer systems with electronic ignitions and fuel injection, there were a handful of things that you could adjust to stop a car from idling roughly. Oracle has far too many parameters to adjust for you to jump under the hood and start turning screws. You need to have a picture of what the problems with the current system are before you can start making modifications.

A good place to start is the tuning report discussed in Chapter 30. This report is designed to capture the parameters that are most likely to show up as a tuning problem within the Oracle database. True, there are hundreds of parameters that this report does not cover, but I could not find any “approved” limits for these other parameters and have not found them to cause problems in the instances on which I have worked.

The tuning report is a good place to start because it divides the world of problems into domains (see Figure 32.2). This report is designed to determine common problems with the way Oracle itself is set up. It detects overloaded disk drives, improper Oracle memory area sizing, and even plain old overloaded computers. (I guess that I prefer not to open my mouth about a problem before I am sure that it

is not something that I did.) When people say that it must be an Oracle problem, you can bring in the references (the Server Administrator Guide and Oracle tuning book for the host computer) that show that Oracle is saying that the system is performing up to its capabilities.

Figure 32.2.
Dividing performance
problems.



Once you clarify where the problem is, you can start working on an appropriate course of action. If Oracle tuning is required, you can go into the initialization files or plan tablespace reorganizations to correct the problems. If the problem lies with overloaded operating systems, it is time to investigate operating system tuning or CPU capacity upgrades. Finally, as is often the case, if the problem is that the applications are being too demanding, it is time to look at tuning the applications to make them more efficient.

The goal of this process is to perform tuning in an intelligent, controlled manner. Altering Oracle system parameters can cause some serious problems if you do not know what you are doing. You can easily make things very bad by adjusting the parameters incorrectly. In addition, because many of the parameters are designed to prevent you from reaching a limit, if you adjust the wrong parameter, you may waste a lot of memory and not see any differences in performance. The only way to reduce memory usage is to decrease the tuning parameters until you see performance degradation. This can take a lot of time, which you usually do not have on production systems.

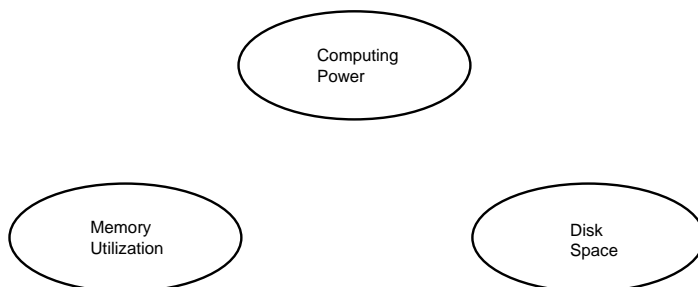
WHAT CAN YOU CONTROL?

Two good philosophies to consider when you are working on tuning your Oracle instance are “Don’t sweat the things that you cannot do anything about” and “Don’t sweat the small stuff.” Let’s start with the first one. There are many things about the way Oracle does things that you cannot control. Perhaps you would like to parallel query processes with your Oracle 6 database. Perhaps you would like a database that is less memory-intensive. Sorry, it just does not work that way. You could go to another vendor’s products, but they are all pretty similar when viewed in an overall perspective (they all have their problems).

Next, let's address the small stuff. There are hundreds of Oracle parameters and ways that you could write your applications that may give you a small performance improvement. If you have a lot of time on your hands, perhaps you can conduct some experiments and publish the results for the rest of us. However, in the typical production database world, your time is limited, so you have to look at the things that are most likely to help you out. That is what the tuning report (which was derived from a number of sources, including the Oracle documentation and Compuserve bulletin boards) is designed to capture for you—the parameters that are likely to give you the most bang for the buck.

With all this in mind, let's look at the factors that you can and should consider controlling to improve the performance of your system. In any computer performance tuning exercise, you usually break the world down into three basic resources—computational power, memory space, and disk space. This is a good high-level start at understanding where the problems lie within your computer system. Typically, operating systems provide resources (monitor under VMS or `sar` under UNIX) that paint a picture of the high-level usage of these resources. When you have a tunable resource, such as Oracle, and locally developed applications that can be optimized, it is important to break down these general resource categories into smaller components that can then be investigated (see Figure 32.3).

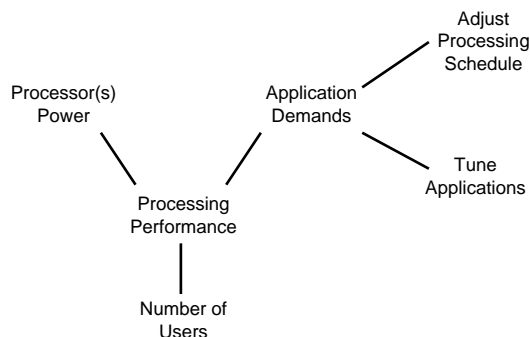
Figure 32.3.
Resources in tuning a
computer system with
Oracle.



The first area to consider is computing power (see Figure 32.4). This is a function of the power of the processor(s), the demands of the applications, and the number of users on the system. Because the number of users is a fixed demand (computers are purchased to serve users), consider that as a given. This leaves the power of the processor or processors and the demands of the applications as your only two variables. Upgrading the processing power is usually possible through purchasing computer system upgrades; however, being one of those capital expenses that everyone scrutinizes, it is usually best to do other things to lower the processing demand. This leaves the demands of the applications as the first variable that you can try to control. The easiest thing that you can do is adjust the processing cycle to level demand throughout the day, week, and so forth. This often involves moving batch jobs around or coordinating people's schedules for work on the system. The other factor that you can control is how much load the applications put on the

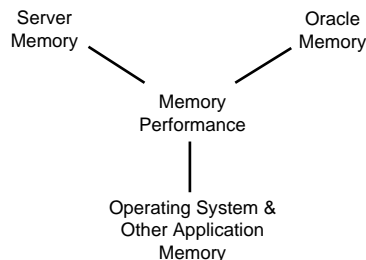
computer when performing their tasks. This is application tuning, and it can provide some remarkable reductions in the loading on a computer with the slightest of changes.

Figure 32.4.
Processing capacity
variables.



The next area of the computer to consider is the memory utilization. This is an especially severe problem in database servers because they all tend to use a lot of memory to improve performance. Once again, break the problem down into pieces so that things can be attacked in some kind of order (see Figure 32.5). The first variable is the total amount of memory available on your server. You may wind up having to add more memory, but you usually have to prove that you have done everything possible to avoid it. The first option is to control the amount of space used by the Oracle instance itself (the SGA). The next option is to control the amount of memory space used by the various user processes. Finally, you have to look at controlling the memory space used by the operating system and non-Oracle applications.

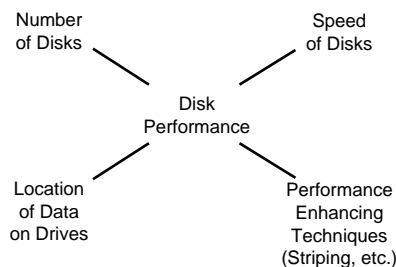
Figure 32.5.
Memory capacity
variables.



The final area in the tuning world is disk space. Once again, you start with the number and type of disk drives installed on your system. This determines the amount of space and how quickly an individual set of data on disk can be accessed. You also can control the location of data on the disk drives to balance the input/output load of the computer and thereby increase performance. Additionally, there

are performance enhancing techniques, such as disk striping, that you may want to consider (more about disk striping soon). Figure 32.6 illustrates the options in this area.

Figure 32.6.
Options in the disk
storage arena.



For now, let's concentrate on the things that you have control over within the instance that will be the basis for these tuning discussions. (Chapter 33 discusses upgrading the computer hardware, because it does not fall directly under tuning—it's more like replacing the car's engine rather than adjusting the spark plugs.) The following items fall into the category of tuning:

- ◆ Adjusting the demand of the application
- ◆ Adjusting the memory space used by Oracle
- ◆ Adjusting the memory space used by Oracle user processes
- ◆ Adjusting the location of the data
- ◆ Special techniques for data storage, such as striping

The first subject to consider is adjusting the demand of the applications for CPU processing capacity. You can think of this as trying to live within your means. There are two primary techniques that can be used to solve this problem. The first involves making the applications more efficient in their use of CPU resources. When you seek to optimize the applications, you need to understand that it is not that the programmers did their job incorrectly. Instead, you have to look into the mystical internals of Oracle to understand how Oracle is thinking and which of the internal algorithms it is using to process their requests. You then work with these internals to find a method that has the best performance (Oracle is working on its optimizer to make it more intelligent with each release).

The other technique that can be used to solve computer processing capacity problems is adjusting when jobs are executed. The vast majority of computer installations have peak times and dead times. Computers typically get fed electricity twenty-four hours a day, seven days a week. Other than backups and rare preventive maintenance, they do not require rest. Therefore, if you can just shift jobs that are flexible from peak periods to the off periods, you can get more work out of

your existing computer. Sometimes it is as simple as posting a notice to the users explaining that the system is overloaded from 8–9 a.m. and 1–2 p.m. Many will want to avoid the long processing delays during these periods and rearrange their schedules to get at the system when performance is best. You should also look at your scheduled and batch jobs to ensure that you are not competing for processing cycles during these busy periods. Always try to run batch reports during the evening or night hours.

The next topic that falls under tuning is adjusting the memory used by your Oracle instances. The biggest impact comes from adjusting the sizes of the SGA and PGA. You control these items using parameters that are in the Oracle initialization files (`init.ora` and `config.ora`). This is not always something that you will be trying to reduce. The vast majority of memory tuning that I have done involves adding memory to one of the SGA or PGA areas to improve performance. The key to this area is that you need to have some data (the tuning report) that indicates which parameter needs to be adjusted. Unfortunately, although the tuning report will show a need for additional memory for the database buffer cache, for example, it will not show how much memory is needed. Therefore, you may have to make several small adjustments over a period of several days to find the value of the initialization parameter that works best for your needs.

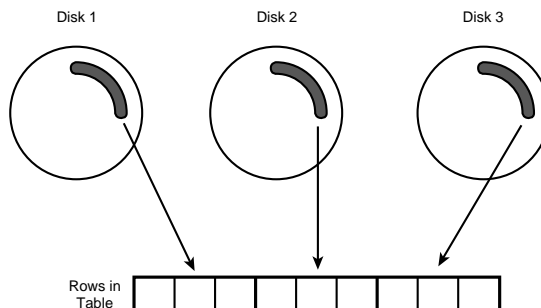
The memory of user processes is the next tuning task that you might need to perform. The developers may be able to do something when they design their applications so that it uses memory overlays or divides the application into a number of smaller modules that consume less memory. If you are using client-server applications, you can try SQL*Net version 2, which enables you to use the multi-threaded server architecture. This will alleviate the need to have a dedicated server process for each user and instead will share a pool of server and dispatcher processes.

The most common tuning task that I have had to perform on new databases is balancing the input/output load across multiple disk drives. All too often, tablespaces are designed so that data is stored according to some form of simple logic, such as all the lookup tables are grouped together. Without any actual performance data or modeling, it is hard to predict what the loading will be like (it depends on the frequency of the query, the execution plan selected by Oracle, and so forth). Therefore, you should look at the loading of your various disk drives to see whether they are impacting the performance of your applications and the system. Your task then is to move “hot” tables onto separate disk drives so that the disk loading is balanced. You may also want to consider consolidating small tablespaces that do not have significant input/output rates onto the same disk drive so that you have more disks free when you need to relocate the heavily loaded tables.

The final tuning topic that you may run across involves special data storage techniques that can increase your performance. This is where the system types can baffle many DBAs with a barrage of terms such as Raid 0+1, striping, and mirroring. There are two key items that you can use as a DBA to improve performance—striping and higher throughput storage devices. Let's look at disk striping first.

Disk striping is a technique wherein a file is scattered across several disk drives so that the input/output load is distributed across the disk drives (see Figure 32.7). This is a useful technique when you have a few large files that place a heavy input/output load on their disk drive by themselves. Because you cannot divide the table across multiple disk drives within Oracle, a technique such as dividing the data file across multiple disks but having it look as though it were a single file on a single disk drive is very useful. There are multiple forms of disk striping, and someone has to worry about details such as the size of the stripe. Work with your system administrator to set up a striping program that meets your needs, given your hardware and operating system environment.

Figure 32.7.
Disk striping concepts.

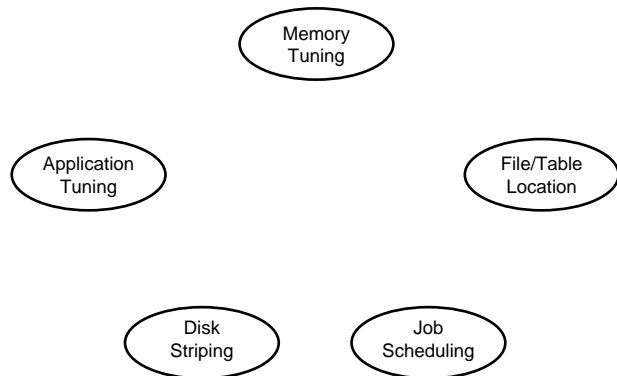


The other special storage technique that can be used to improve data transfer performance falls into the general topic of *hierarchical storage*. Originally, this term was used to describe mainframe systems wherein data was transferred from disk drives to lower access speed magnetic tapes. However, many vendors have extended these concepts in the other direction to include systems that provide higher input/output performance than the regular disk drives. This may include disk drives that are optimized for throughput (as opposed to cost or capacity). It may also include the use of memory to simulate a disk drive but with much higher performance. These are tools that can be useful when you have some tables that are used heavily but are too large to load into memory.

This section has covered the parameters that you can control to improve the performance of the Oracle instance. There are a number of performance improvement techniques, such as buying a new computer, which do not fall under the heading of tuning. Instead, tuning can be accomplished with a handful of steps and

can usually be executed without major purchases or reduction of service to the users (see Figure 32.8). Again, there are many Oracle parameters and system techniques that may improve the performance of your Oracle applications. These are the ones that are recommended in the standard Oracle literature and apply to most of the situations that I have run across. If you run regular tuning reports and can optimize your system with these parameters, you will probably be ahead of over 90 percent of the DBAs out there.

Figure 32.8.
Summary of Oracle
tuning techniques.



HOST COMPUTER INDICATORS

How do you know what to adjust on your computer to improve performance? Most computer operating systems provide utilities that measure the load from the whole system perspective. It is quite possible that you could have an Oracle application that is performing poorly on a computer system that is lightly loaded. This could happen when you have an application that is poorly written or your Oracle memory areas are not sized properly. You could also have the situation where Oracle is well-tuned and your applications are well-designed, but your computer system is just overloaded.

There are three key resources that most operating systems can measure for you—CPU capacity, disk input/output, and memory utilization. Under UNIX, you use `sar`, `vmstat`, or `iostat` to measure these parameters. Under VMS, you use the monitoring utility. The best thing for a DBA to do to understand these utilities is to get with your system administrator. You may need to be granted permission to access these tools, and different versions of the operating system may have different interpretations for results.

Let's start with CPU utilization. Typically, you will see three components of CPU utilization on your monitoring utility. The first lists the utilization by the system (%sys in `sar` under UNIX). This usually is not a problem for the Oracle DBA to correct (you have to let the operating system administrator have some of the fun). The next

column is the utilization by the user processes (%usr in sar). Typically, this is where you want to spend your processing time serving user needs. The third component of interest is the time spent waiting for input/output operations to occur (%wio). This is where you do not want to spend a lot of time (it is a sign that you are having input/output problems and you are wasting CPU time waiting around).

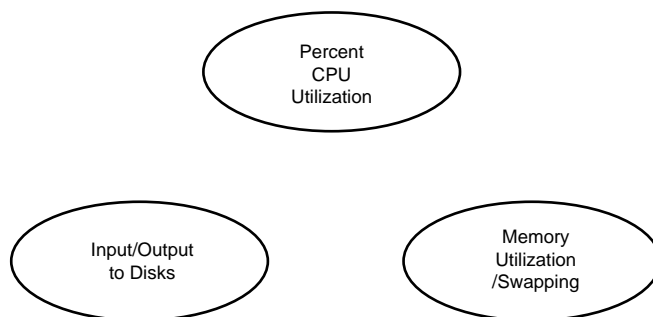
When do these numbers indicate a problem for you? First, look at the sum of these numbers (or, under sar, 100 percent minus these numbers, which shows idle time). If you are consistently running at or near 100 percent utilization (%idle) and you have response time problems, you may have an overloaded CPU. The word “may” was used because there might be another problem. Remember to check the percentage of time spent waiting for input/output operations to occur. If you have a very high %wio, the real problem may be input/output load balancing. I have seen systems that run with 0 percent idle until a few tables are moved around and then suddenly the percent idle goes to a very reasonable 50 percent value. However, if you are running flat out and it is consumed with a balance of user, system, and input/output operations, you probably have an overloaded CPU. If so, you have to consider shifting jobs to off-peak periods, tuning the applications to be less demanding, or acquiring additional CPU capacity.

As mentioned earlier, CPU utilization problems can crop up due to disk utilization problems. However, you can have input/output loading problems without their showing up on the CPU utilization monitors—especially in multiprocessor systems where a process waiting for input/output would tie up only one of the many processors, thereby using a small amount of the total processing capacity. There are separate monitors that show the number of input/output operations to each of the disk drives. The transfer capacity of the disk drives varies; therefore, you should talk to your system administrator to find the appropriate value for your system. Once you have this number, you have to check your actual load against this expected number to ensure that you are not exceeding the maximum transfer rate and thereby overloading a disk drive. If you find that you have disks that are excessively active, your options include moving tables around to level the load and special data storage techniques, such as striping.

The final operating system monitoring utility that you need to look at on a routine basis is memory utilization. This can be an absolute killer on database systems, and it may well be difficult to see unless you know what you are seeking. This goes back to the concept of virtual memory. When the capacity of the physical memory is exceeded, data is transferred from physical memory to disk drives. When it is needed by the programs that are executing, it has to be transferred back into memory. This swapping can severely degrade the performance of your system. Therefore, you need to keep an eye on how much of the total system memory is being utilized (which is sometimes not clearly indicated) and whether you are swapping or not. Again, it is best to get with your system administrator to go over your monitoring utilities and determine what to look for on your operating system.

This concludes the quick and dirty discussion of monitoring the overall loading of your computer system. There are three key resources that are typically monitored at the operating system level (see Figure 32.9). There may be some environments wherein the systems folks will get upset if the DBAs try to monitor what is under their purview. Some of you may prefer to let them deal with the intricacies of the operating system and let you worry about Oracle. Part of my fondness for using these utilities is that I have also worked as a system administrator and am therefore comfortable with using them. It is important that you have access to some data that indicates whether the problem is with Oracle or with the system as a whole. You may have a perfectly tuned Oracle instance that performs poorly because the CPU is just too small. You can also have a system that is supporting multiple projects wherein other applications may be overloading the system along with Oracle. Finally, you may honestly have a tuning problem with Oracle that you need to solve. The data from the operating system will help you determine where the problem lies.

Figure 32.9.
Resources monitored at
the operating system
level.



ORACLE RESOURCE CONTENTION

There is a possibility that performance problems are caused by Oracle being out of tune. The operating system monitoring utilities may point to a particular problem (such as disk input/output contention), or they may not. Your job then is to look inside Oracle and see whether there is anything that you can do to make things better. Again, a good basis for judging the soundness of the tuning of the Oracle instance is the tuning report. This captures the recommendations in the Server Administrator Guide, the Oracle tuning references and some wisdom from bulletin boards such as those on Compuserve. Taken together, these analyses capture the list of parameters that in most cases reflect how well-tuned your Oracle instance is.

This section discusses these Oracle resources and presents the corrective actions that I typically consider. One of the more challenging parts of tuning is that you usually get data that indicates that a parameter needs to be changed—but not by how much. An example might be that you find a high percentage of your sorts being performed on disk as opposed to in memory. You don't know whether the sorts that

are causing this problem need just another kilobyte of sort space or whether these are extremely large sorts that would take gigabytes of physical memory and therefore will never fit in memory. The solution is to adjust the parameters gradually in the right direction and then look at the results in your monitoring program. But first, let's look at a list of the parameters that I typically monitor to assess the tuning of an instance:

- ◆ Library cache miss rate
- ◆ Data dictionary cache miss rate
- ◆ Multi-threaded server session memory versus shared pool size
- ◆ Buffer cache hit rate
- ◆ Disk activity (by Oracle data file)
- ◆ Rollback segment contention
- ◆ Latch contention
- ◆ Multi-threaded server dispatcher contention
- ◆ Shared server process contention
- ◆ Redo log buffer space contention
- ◆ Sort memory contention
- ◆ Free list contention

The first item that I check for is the library cache miss rate. This measures Oracle's capability of reusing parsed SQL statements, which is a way to increase Oracle's performance. Your goal here is to miss on repeated statements less than one percent of the time. If you find that you are exceeding this limit, you should increase the `shared_pool_size` parameter in your `init_sid.ora` file. The following excerpt from one of my instances shows a properly tuned library cache:

```
#####

Library Cache Check

Goal:                                <1%

Corrective Action:                   Increase shared_pool_size.
                                     Write identical SQL statements

Library Cache Miss Ratio (%)
-----
                                     .33

#####
```

The next parameter that needs to be monitored is the data dictionary cache miss rate. The data dictionary cache is used to store data about the structure of your

database in memory and thereby speed up processing of statements. You will have no information in the data dictionary cache when you start your instance; however, you should build up this information as people access the instance. The goal in this situation is to have a miss rate of less than 10 percent. If you are not achieving this goal, you should increase the `shared_pool_size` parameter in your `init_sid.ora` file. The following example of a data dictionary cache check shows an instance that is exceeding the recommended goal (I had just started the instance up, therefore this is not an actual problem):

```
#####
```

```
Data Dictionary Cache Check
```

```
Goal:                                <10%
```

```
Corrective Actions:      Increase shared_pool_size
```

```
Data Dictionary Cache Miss Ratio (%)
```

```
-----  
                                17.09
```

```
#####
```

Another indication that the shared pool may not be large enough comes from a check of the multi-threaded server session memory. It may seem strange that you need to look at multiple indicators to determine whether a single parameter is not correctly sized, but you have to remember that different applications stress different components within Oracle. Therefore, you have to look at all the possible symptoms to detect a problem. The multi-threaded server is that feature of Oracle version 7 that works with SQL*Net version 2 and enables you to have the users share a series of server processes rather than require a dedicated process for each user. The feature that you are measuring is the maximum amount of space used by user processes (session memory) against the shared pool size. Your goal is to have the shared pool at least as large as the maximum session memory. If you do not achieve this, you need to increase the `shared_pool_size` parameter. The following is an example from the tuning report that shows an instance that does not have a problem:

```
#####
```

```
Multi-Threaded Server Session Memory
```

```
Goal:                                Shared_pool_size at lease equal to maximum.  
                                session memory
```

```
Corrective Action:      Increase shared_pool_size
```

Session Memory (Bytes)

```
-----
                25,000
```

Shared_Pool_Size (Bytes)

```
-----
            3,500,000
```

#####

The next tuning item to check is the buffer cache hit ratio. Notice that the previous items were miss ratios and this is a hit ratio. It would be nice to be consistent (always measure misses), but you have to live with what Oracle is recording internally so this is a hit ratio. The buffer cache is used to buffer data on its way to or from the data files themselves. Your goal here is to have a hit ratio above 60–70 percent. If you are not achieving this, try increasing the `db_block_buffers` parameter. The following is an example of a small instance that is doing very well at achieving this goal:

#####

Buffer Cache Hit Ratio

Goal: Above 60 to 70 percent

Corrective Action: Increase `db_block_buffers`

Hit Ratio (%)

```
-----
            92.0
```

#####

Now let's shift focus briefing from internal Oracle tuning parameters to disk input/output. If you have a small database, you may have to keep data files for multiple tablespaces on the same disk drive. Therefore, when you detect disk contention at the operating system level, you may not be sure where the culprit is that is causing this problem. Oracle provides you with the capability of monitoring input/output activity on a data-file-by-data-file basis. Your goal here is to have the load balanced between disks (not just balanced between data files). If you detect problems, you need to find ways to move tables or indexes between data files (tablespaces) to balance the load. The following example shows an instance wherein almost all the disk activity is concentrated on one data file and disk, but the load is so light that this is not a problem:

#####

Disk Activity Check

Goal: Balance Load Between Disks

Corrective Action: Transfer files, reduce other loads to disks,
 striping disks, separating data files and redo
 logs.

Data File	Reads	Writes
C:\ORAWIN\DBS\wdbsys.ora	268	
C:\ORAWIN\DBS\wdbuser.ora		
C:\ORAWIN\DBS\wdbrrbs.ora	2	
C:\ORAWIN\DBS\wdbtemp.ora		

#####

One of the more confusing and difficult-to-manage internal Oracle structures is the rollback segment (used to store transactions that are in progress). You have a fixed number of these objects that typically reside in their own tablespace. Your goal is to keep contentions less than one percent of the total “get” attempts. If you do not achieve this goal, you should consider adding additional rollback segments using the `create public rollback segment` command. If you are not familiar or comfortable with rollback segments, Chapter 42 contains some hints about the unique storage parameters and “gotchas” associated with this type of object. The following example shows the monitoring report for an instance that has no problems with its rollback segments:

#####

Rollback Segment Contention

Goal: Measured Counts < 1% of total gets
 (the choice of Oracle column names makes it
 impossible to do this calculation for you)

Corrective Action: Add more rollback segments

 Total Gets

 5,739

Class	Counts
system undo header	
system undo block	
undo header	
undo block	

#####

Next on your list of tuning problems is latch contention. Latches are an internal Oracle construct that controls access to the redo log buffer. You probably do not need to get into the details of how these things work. You just have to know that if you have more than a one percent miss rate, it may be time to adjust the `log_small_entry_max_size` or `log_simultaneous_copies` parameter in your initialization files. Remember, if you cannot find these parameters in your initialization files, you are accepting the defaults for your operating system. To change this, all you have to do is add the parameter with a value that is greater than that of the operating system default to the initialization file. If you are unsure what the default for your operating system is, you can get the current value from the configuration report. The following is an example from my Personal Oracle7 instance running under Microsoft Windows that shows no problems:

```
#####
```

Latch Contention Analysis

```
Goal:                < 1% miss/get for redo allocation
                   < 1% immediate miss/get for redo copy
```

```
Corrective Action:   Redo allocation-  decrease log_small_entry_
                   max_size
                   Redo copy-  Increase log_simultaneous_copies
```

Latch Type	Misses/Gets (%)	Immediate Misses/Gets (%)
redo allocation	.00000	.00000
redo copy	.00000	.00000

```
#####
```

If you are using the multi-threaded server, you could have contention for dispatcher processes. This occurs when a user tries to connect to a dispatcher to get access to a shared server, but cannot because the dispatchers are busy. The recommended goal is to have this happen less than 50 percent of the time. If it occurs more than 50 percent of the time, you should alter the `MTS_MAX_DISPATCHERS` parameter in your initialization file.

Related to the multi-threaded server dispatcher contention issue is contention for multi-threaded server processes. Your goal is to keep the multi-threaded server processes greater than the number needed. These processes will be killed if they are not needed, so you are not wasting system resources by having a high value for this parameter. If you need to add server processes, you should increase the `MTS_MAX_SERVERS` parameter in your initialization file.

Next, let's consider redo log buffer space contention. The redo log stores transactions that are waiting to be written to the redo log files. Because this involves Oracle's

capability of recovering the database in the event of failure, Oracle is very strict about not overwriting data. Therefore, your process will sit and wait (increasing response time) until the log writer has cleared out some space for you in the redo log buffer. Your goal therefore is to keep contention for redo log buffer space near zero (if you have millions of transactions, a few contentions will not kill you). To correct a problem, you need to increase the `redo_log_buffer` parameter in your initialization file. The following is an example of an instance that is not having any problems with redo log buffer space:

```
#####
```

```
Redo Log Buffer Space Contention
```

```
Goal:                               Near 0
```

```
Corrective Action:                   Increase size of redo log buffer
```

```
Requests
-----
```

```
#####
```

There are two tuning parameters left. The first is the sort memory availability. Because most disks are at least an order of magnitude slower than physical computer memory and most database applications present data sorted in some logical order, this can be an important performance issue. However, you may also have large data warehouses that bring down enormous amounts of data that are too much for the physical memory that can be reasonably installed on your system. Therefore, you need to check whether your system is sorting to disk or in memory and see whether you can set a reasonable amount of space in memory that will cause these sorts to be performed in memory. If you need more sort memory, you should increase the `sort_area_size` parameter in the initialization files. The following is an example of an instance that is in good shape with regard to sorts:

```
#####
```

```
Sort Memory Contention
```

```
Goal:                               Mimimize sorts to disk
```

```
Corrective Action:                   Increase sort-area-size
```

```
Type                               Number
-----
```

```
sorts (memory)      148
sorts (disk)
```

```
#####
```

The final tuning parameter that I routinely monitor involves the free list. The free list is the list of available blocks within an extent. I have never had to set the free list storage parameter for tables. However, if you find a problem with this monitoring parameter, you have to figure out which table is causing you the problem and then re-create that table (with perhaps an export and then import of data) with a larger value for the free list. Let's hope that you never have to deal with this problem.

There are plenty of other parameters. This has been a discussion of the issues that are likely to arise. There are entire books and third-party monitoring tools that deal with these rarer issues. However, I usually feel confident that my database is working well if it passes these checks. You may find, when working with special types of Oracle instances (for example, huge data warehouses), that there are some undocumented Oracle parameters that Oracle may recommend you use. Some of these may help the performance of your instance; however, you are dependent on Oracle to suggest them. If you are tuned according to these checks, you will probably be ahead of 90 percent of the DBAs working today.

Before we leave this discussion, let's look at a reference on all of the documented tuning parameters in Oracle. My favorite is Appendix A of the *Server Administrator Guide*. In there, you will find the parameters and a brief description of each of them. Do not adjust a parameter until you are sure of what you are doing, even if someone else advises that changing it could help.

A TUNING CHECKLIST

The following is a quick checklist showing the thought processes that I typically go through when tuning an Oracle instance. You should look at all the parameters in the tuning report before touching anything. You should check out the real end goal of performance tuning: response time in the instance. Always make a copy of the initialization files before you edit them. My favorite approach is to copy the file to a file that contains the current date (for example, copy `init_test.ora` to `init_test.ora.051295`). This helps because you may change an initialization parameter to a value that does not work for your instance. You could exceed the maximum value allowed for your operating system. There also are a few parameters that insist on being an even multiple of the block size and cause the instance to not come up if they are not set correctly. Finally, you have to recheck your performance tuning after the instance has had some time to run at normal production volumes to see whether your changes have been enough.

CHECKLIST

- ☐ Analyze load on operating system—CPU, disk, and memory.
- ☐ Analyze performance of key Oracle tuning parameters (the tuning report).
- ☐ Analyze the overall response system time (issue a query when you know the expected response time).
 - ☐ Determine adjustments that are needed:
 - ☐ File/table location
 - ☐ Memory tuning
 - ☐ Application tuning
 - ☐ Disk striping and so forth
 - ☐ Job scheduling
- ☐ If altering initialization parameters, make a copy of the current initialization files.
- ☐ Adjust initialization parameters, if needed. Perform any other adjustments needed.
- ☐ Let the system run long enough to obtain representative data and rerun the tuning check. Did your changes accomplish your desired goals? If not, go back to step 1.

SUMMARY

This chapter provides an overview of Oracle performance tuning. The key contrast is that tuning is working with what you have to improve performance. You explored the various resources that you have at your disposal that affect performance. Those that were tuning-related were separated from those items that involved new acquisitions (which is discussed in Chapter 32). Finally, you learned the parameters to monitor and what to adjust to increase performance.

This chapter certainly challenged your knowledge of Oracle internal structures and how they can be used to improve performance. There is a lot more that could be done in this area. There are books, third-party tuning products, and white papers that go into this subject in much more detail. This chapter's goal was to present the survival guide level of knowledge that will solve problems for the vast majority of Oracle DBAs. If you have a relatively small instance, you may never have to worry about tuning. However, if you do have to tune your instance, you can at least feel good that

you are controlling Oracle to get the most out of it and not just throwing more powerful computers or more memory at the problem. You can think of yourself as being like a mechanic who keeps vintage 20-year-old cars running smoothly as opposed to the guy who just does oil changes.



- Knowing Where the Database Is Going
- When More Is Needed
- Proving Your Case

CHAPTER 33



Looking Toward the Future

So far in this part of the book, you have learned a number of ways to measure and adjust the health of your database. Before leaving this section, let's take some of these proactive management concepts to the next level. Rather than worrying about when you will need to expand your tablespace, consider planning the amount of disk space that you will need next year. That is what this chapter is about. It is a process of taking the data that you have been so diligently collecting with your monitoring program and putting it to use for future planning.

This chapter approaches this topic first with a discussion of how to know where your database is going. This involves some interaction with the users and developers combined with experience at sorting wild ideas from firm plans. Next, you learn when more resources are required for your system and some analytical techniques that I have found helpful. Finally, you explore some useful ways to prove your case and get the budgets that you need to keep the system working properly. Again, this is usually difficult because most organizations closely scrutinize capital expenditures (such as computer disk drives).

KNOWING WHERE THE DATABASE IS GOING

You can think of knowing where the database is going as a task comparable to predicting where the stock market will go or whether it will rain on a certain day next month. Sometimes it seems just as challenging. However, it can be done and a forecast that is not perfect is usually better than having no forecast at all. In this section, you learn the two most important points in preparing a forecast of database needs:

- ♦ Historical trends
- ♦ User plans

Both of these components are needed to prepare an accurate forecast.

You already know how to capture the data that you need on the historical trends of your databases. All you have to do is use those monitoring reports and perhaps any usage auditing that you were performing. The key is to keep a trend on the factors that are most important for future planning. First, you need to have a trend over a long enough period of time to be valid. If you took the data over a single month in which you added a lot of data thanks to a new application coming on line, for example, you might conclude that you need an enormous amount of new disk space in the future. You really need many months and perhaps years of data that shows the long-term trend in your disk growth. Then you need to determine the factors that are most important to future planning. You will have a lot of data available to you—disk usage, table fragmentation, and so forth. Your job is to wade through all these numbers to capture those items that are important in the long-term planning process. The following items are a good start at such a list:

- ◆ Disk storage needed for data, by application
- ◆ Disk storage needed for Oracle and application software
- ◆ Processing capacity utilization
- ◆ Memory utilization for Oracle instances
- ◆ Overall system memory utilization
- ◆ Number of people using the system

Most of this data is available from the various Oracle reports discussed in Chapter 30. You merely store it for analysis at a later date. My favorite method involves using computer spreadsheets to store the information, because I can easily produce graphs, averages, and so forth later. However, as with so many things these days, information overload can get to you. Therefore, let's spend some time exploring how to select data for a long-term monitoring program.

If you keep a nauseatingly detailed day-by-day account of how much space every table within Oracle is consuming, you will probably have trouble getting the rest of your work done. If you keep only a total value for disk storage utilized on your server as of January 1 of each year, you will probably lack the detail needed to make a convincing capacity analysis. The best level of detail lies somewhere between, but where? Once again, you have to pick the level of detail that is right for your situation. Some places neither want nor require extremely detailed projections, and others demand them.

Having said all that, let's look at the level of detail to keep for an Oracle instance and database. An excellent level of detail for the time period is a single month at a time. This is usually good enough to draw a line through to show general trends. Make it a point to highlight any special events that occur during a given month (for example, a new system comes online) so that you can understand any jumps in the curve. You can capture the information listed previously with the following levels of detail:

- ◆ Disk storage for the Oracle data is captured on a tablespace-by-tablespace basis. If you have most of your data in a few big tables and the rest merely serve as references, you may want to capture the size of these tables also.
- ◆ Disk storage for the Oracle software typically changes only when you upgrade the software or add new modules. It is typically small relative to the size of most databases. However, it is important to keep track of so that you are sure you have room for it.
- ◆ For processing utilization, try to estimate an average value for the month. This data has to be taken when there is a representative load on the system from the users. Record the following values on a UNIX box: %sys (system utilization), %wio (waiting for input/output), and %usr (user utilization). You do not need to try and take this to three decimal places; just get a number that is reasonably close. Throw out numbers if you have a nonrepresentative data sample (your report ran when your business was closed).

- ◆ Use the `show sga` command to capture the amount of memory space used by Oracle, again on a monthly basis. The Oracle background does take up space, but this is accounted for in the overall memory utilization figure that is mentioned in the next bullet point.
- ◆ The overall memory utilization comes from the standard tuning report that you run. Again, the goal is to get a representative average for the month. This system level report (`sar` under UNIX) shows all memory utilization activities.
- ◆ Finally, an estimate of the number of people using the system for the month can be helpful. There are two ways of looking at this. The simplest number to obtain is a count of the number of users with login IDs (select `username` from `dba_users` and count the ones that match up to real end users). Another alternative is to run several counts during the day of the number of users actually logged on to the system (`who` under UNIX) or who are accessing Oracle (using `monitor users` under `SQL*DBA`). The latter option provides a better number in client-server environments.

So far, you have captured a lesson in the history of your system. You will be able to show overall trends in data and usage growth. However, this is adequate for predicting future needs only if there are no changes to the system, which is a rare occurrence. Therefore, you need to keep in touch with the users and developers to see what they have planned for the future. In some locations, you may have formal procedures that require them to get approval for additional disk drives to support new data requirements. If not (or if the numbers they come up with are notoriously low), you probably need to track all upcoming projects and estimate their impacts on the system.

This estimation of the impact of future projects can be difficult. How can you size an application that has not been designed yet? A usual practice is to make a series of estimates as time goes on, each one using a slightly more accurate method. For example, you may start out with only a rough concept that the future project will add data for another plant into your application. You may estimate that this will double the size of your data tables. As the design progresses, you find that this plant does only half the business of your existing plant; therefore, you revise your estimates downward. Finally, as implementation time approaches, you perform a detailed sizing estimate based on the number of rows that will be added to the various tables per month to obtain the amount of disk required to hold five years of data for this plant.

Some estimates may be more difficult. Perhaps you know that there is going to be a sales forecasting instance added to your computer next year. No one has any idea how big it will be. With a lack of information, you may want to go out to the mainframe that is holding the current sales information and find out how much

space the data is consuming in its current flat files. Any time you perform such a rough estimate, you may want to add a safety factor that might vary from 10–50 percent or more.

One item that you may want to put on your “to-do” list once the application development begins is taking some measurements of the data in the test instance once the schema has stabilized. The challenge is to be able to estimate the number of rows in the various tables and indexes when the system grows up to production. You hope the numbers you come up with are close to your earlier estimates. Even if they are not, managers will usually be displeased if they have to hurry and buy extra disks within a month but furious if they are told the application cannot transition to production unless you get 10G of disk by tomorrow.

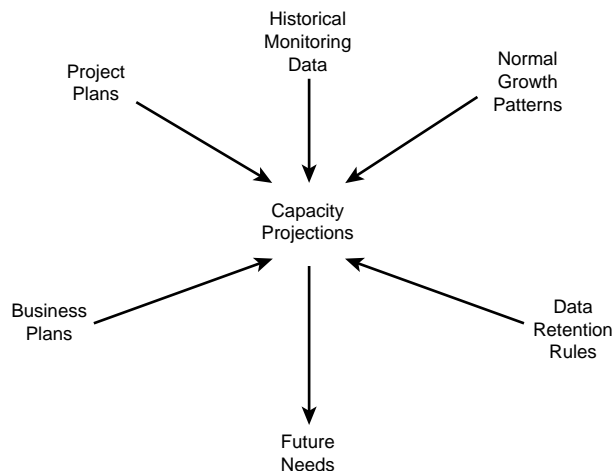
New projects may not be the only business factors that you need to take into account when you are gazing through your crystal ball. They are the ones that tend to appear on official schedules and therefore are often the easiest to track down. However, you should at least consider some of the following:

- ◆ *Changes in data retention.* Users may want to keep more years of data on line for comparison reports. As a system goes along, you also may find that you reach the limit for the number of years of data that you are required to store, and therefore demand will level out somewhat.
- ◆ *A change in the volume of business that the system is being asked to support.* If the instance is supporting a new product line or business, you may have only a handful of clients during the first year, but add additional clients in time. Your challenge is to tie the volume of space that you need to the business units projections of workload. For example, if the business unit estimates that you will be processing 10 claims per day in the first year and 1,000 claims per day in the second year, you are justified in expecting that the disk storage space will increase by a factor of 100 in the second year. It is important to show on your estimates that you are using the business units numbers as the basis for your projections (they tend to argue less that way).
- ◆ *Situations in which you need scratch space.* Always have a few extra disks that can serve as hot spares to recover quickly from a disk crash. This can also be important during some installation and conversion efforts. For example, you may want to make a copy of your existing data tables, build the new tables, and then convert the old data to the new format. Scratch disks are a nice place to put this conversion tablespace that will go away as soon as the conversion is completed.

Because this is a DBA survival guide, not a capacity planning book, this section can be summarized by suggesting that you should capture the data from your routine monitoring program for future reference. You need to select a reasonable sample of

that data and key parameters that will help you track the things that you are likely to need more of in the future—processing capacity, disk space, and memory. Once you have this data, the job is only half complete. Although you have an accurate picture of what has happened in the past, you still need some data about changes planned for the future. You need to go to users, developers, managers, and all others in your organization who understand what is planned for the future. You need to take the plans and estimate the impacts on your system. The estimates may be inaccurate at first, but do the best that you can and refine the estimates as you go along. Your goal is to be able to produce a picture of what resources are going to be required in the future for your database (see Figure 33.1). The time scale of your projections is usually tied to budget periods such as fiscal years, months, and so forth.

Figure 33.1.
Future resource
projections.



WHEN MORE IS NEEDED

By now, you have a wonderful set of projections as to what you will need in terms of disk, memory, and computing capacity. Your numbers are as accurate as possible and show a great amount of diligence and effort on your part. Now for the big question: What do all these numbers mean?

Your next task is to turn these numbers into either an assurance to your management that your existing equipment can meet your needs or a shopping list of components that will be needed. The first thing that you need to make this assessment is a listing of your current capacities. This starts with list of disk drives, processors, and memory modules. The first decision call that you have to make is when do you reach your limit. This is not easy because there are a number of inputs to the process:

- ◆ Vendor recommendations as to the point where performance will degrade (for example, when %idle is less than 20 percent, performance will drop off).
- ◆ Local experience as to what you and your users consider adequate performance (perhaps they start to complain when %idle reaches 30 percent).
- ◆ Margins of error factored into projects to account for the fact that your estimates are based on a number of unknowns. Sometimes you are allowed to put these numbers directly into your presentations. Perhaps you figure out your disk space and then add a line item for margin of error at, for example, 20 percent. In other organizations, people expect absolute clairvoyance, and therefore you have to hide safety margins in your estimates.
- ◆ Effects of routine maintenance such as upgrading your operating system or Oracle database software. This is not usually considered and planned for as a major project. However, the new software will require more resources than the previous versions of the software. (Remember when DOS ran well on PCs with 128K of RAM?).
- ◆ Some things, such as disk drives, are usually not fully used. For example, if you calculate that you need 6G of disk storage for a system, don't expect to fit it on three 2G disk drives. You will need some extra space and it may impact your ability to move files between disk drives to level input/output loading.

Your goal is to establish a set of limits for your key resources that you can reasonably justify and into which your organization can buy (for example, 20 percent idle for CPU, 70 percent full on disk drives, and 20 percent free memory). Based on this, you can draw out a set of curves that will show you when you need to increase your capacity (these are discussed in more detail in the next section). If the answer is that you do not need to increase capacity, you will probably be quite popular with your management (unless you make a mistake and it turns into a crisis later). However, so far you have a sound set of estimates for the requirements of your system and a reasonable estimate of your current resources. The sales job is saved for the next section.

PROVING YOUR CASE

Perhaps you have come to the conclusion that you need to increase the capacity of your system in order to keep up with projected demand. You can clearly see it coming. You have 500M of disk space left and you keep adding 200M of data per month. However, management will not just accept your verbal request to purchase the additional 4G of disk that you want. Instead, they want you to write a report or give a briefing to justify your request. That is what this section is about.

Everyone has a different writing or presentation style. The following is the way that I have found has the best chance of succeeding in the range of clients and managers for which I have worked. You may have a boss who has a particular presentation

format that is required or who would hate your particular style. You have to be the judge of that. However, if you are open as to style, here are the recommended steps:

1. Introduce what you are doing and state your goals.
2. Show the historical usage to date.
3. List the assumptions that you are using for future projections, including capacity limits.
4. Show your future projections.
5. State what is needed for the future.
6. Discuss alternatives.

Let's start with the introduction and goals. Too often, computer people who are in the heat of battle forget that others are not as intimate or focused on their particular problems. Therefore, they might assume that everyone knows that this report, for example, is designed to give justifications for additional disk capacity for next fiscal year to accommodate re-hosting three new applications. Begin by stating exactly what the purpose of the report or briefing is, and state it in a positive way that can obtain agreement. Having a purpose of "getting money out of the budget tightwads" is not good. Having a purpose of "ensuring that we will have adequate capacity to support the order entry system" is much nicer and more likely to have the result of everyone agreeing with you at the start of your presentation.

Next, present the historical data showing usage. Again, this is a relatively neutral section that can be defended if questions arise. Assuming your data collection methods were good (you did not collect data points from the middle of the night that showed almost no usage), the data you present is a historical fact and not an opinion. It will usually be a good learning experience for your audience, which is usually not as close to the system as you are.

The next steps will generate a little more in the way of questions. It is time to list the assumptions that you used in preparing your forecasts. This is important because I have sat in on too many briefings where the presenter puts up a chart showing that 20G are needed for next year and the meeting degenerates into a series of questions about whether you took this or that into account. People tend to get confused in this type of setting. Therefore, it's a good idea to lay out your assumptions up front and tie every one of them back to an approved business plan. This might include approved or planned development projects. It should also include business forecasts. (The business types feel good when somebody listens to their sales plans even when you know they'll never reach them.)

You have to be prepared for a couple of tactics at this point if you are giving a presentation as opposed to a report. You may get a manager who starts to question you about details that you are not aware of (such as how the acquisition of this

company would affect the projections). Try to defer questions that are beyond your scope to the appropriate business person who provided the forecast assumptions and point out that the plan could be revised if the business people change their needs. The key is to proceed and not let them tie down your whole presentation over some little detail that they have to work out among themselves.

Another tactic that you may have to deal with is people who are just not sure about anything. One good tactic is to say that this is the best data we have and that the plan could be revised if they get around to deciding (you have to say that politely, of course).

Finally, you might get a person who, with great bravado, says that one of the projects that you were told is upcoming was canceled five minutes ago. Just play right through this problem and say that this will be dealt with when we get to the options part of the presentation.

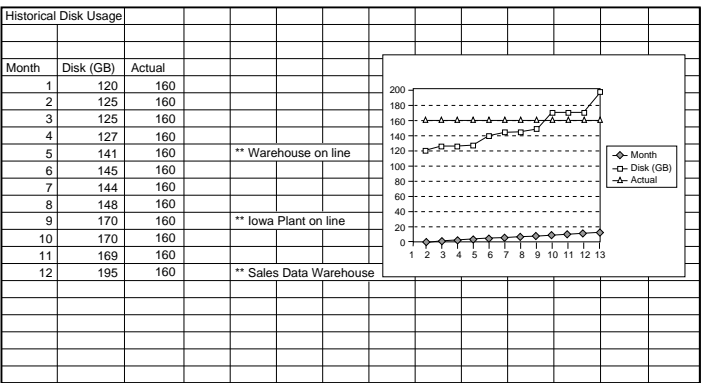
The goal is to survive the assumptions section with only minor cuts and bruises. You have to get used to the fact that there are some people who like to challenge everything in meetings. If you have linked all your assumptions to something that is a measured fact (your historical usage) or a stated business need (an upcoming project or business volume projection), these people will not challenge something that you project. If you have the right people in the room (those who came up with these business projections or are sponsoring the upcoming projects), you can let them present their case. They can usually answer questions about their needs much better than you can.

Now is the time that you have to present your projections. Some places actually prefer a simple slide that says *Needed—2G* and nothing else. Others want a little more detail before they shell out the money. This is where you have to mix presentation style with accurate data. You can, for example, show your projected space requirements for the next six months in a table similar to the following:

<i>Month</i>	<i>Disk (G)</i>
1	120
2	125
3	125
4	127
5	141
6	145
7	144
8	148
9	170
10	170
11	169
12	195

Most audiences tend to get lost and not see that you have to buy 20 2G disk drives by September or you will exceed your current 160G capacity. It is often helpful to build a nice graphical slide that shows major changes and a graphical output that shows where need exceeds capacity (see Figure 33.2). The goal is to make it easy for your audience to get the point that you are trying to make.

Figure 33.2.
Sample slide showing
capacity versus need.



Another important consideration is to give the audience some options. When you are looking into the future, things may be uncertain. If you present only the scenario you are expecting, you may get sent back to the drawing board to tell them what is needed if project xyz is canceled. Therefore, you may want to include a slide similar to the following that shows where your growth is expected:

Project	Disk Space Needed (G)
Sales forecasting growth	5
Warehouse tracking	20
New test instance	10

A lot can be said about presentation options. The keys to focus on include the following:

- ◆ Get agreement on goals early and make sure that they match standard business goals.
- ◆ Present your assumptions up front and tie them to business needs and not just personal guesses.
- ◆ Present a picture of history. It usually shows that computer needs have grown for the last five years. It is much harder for someone to argue that there is no basis for increased capacity after viewing this history.

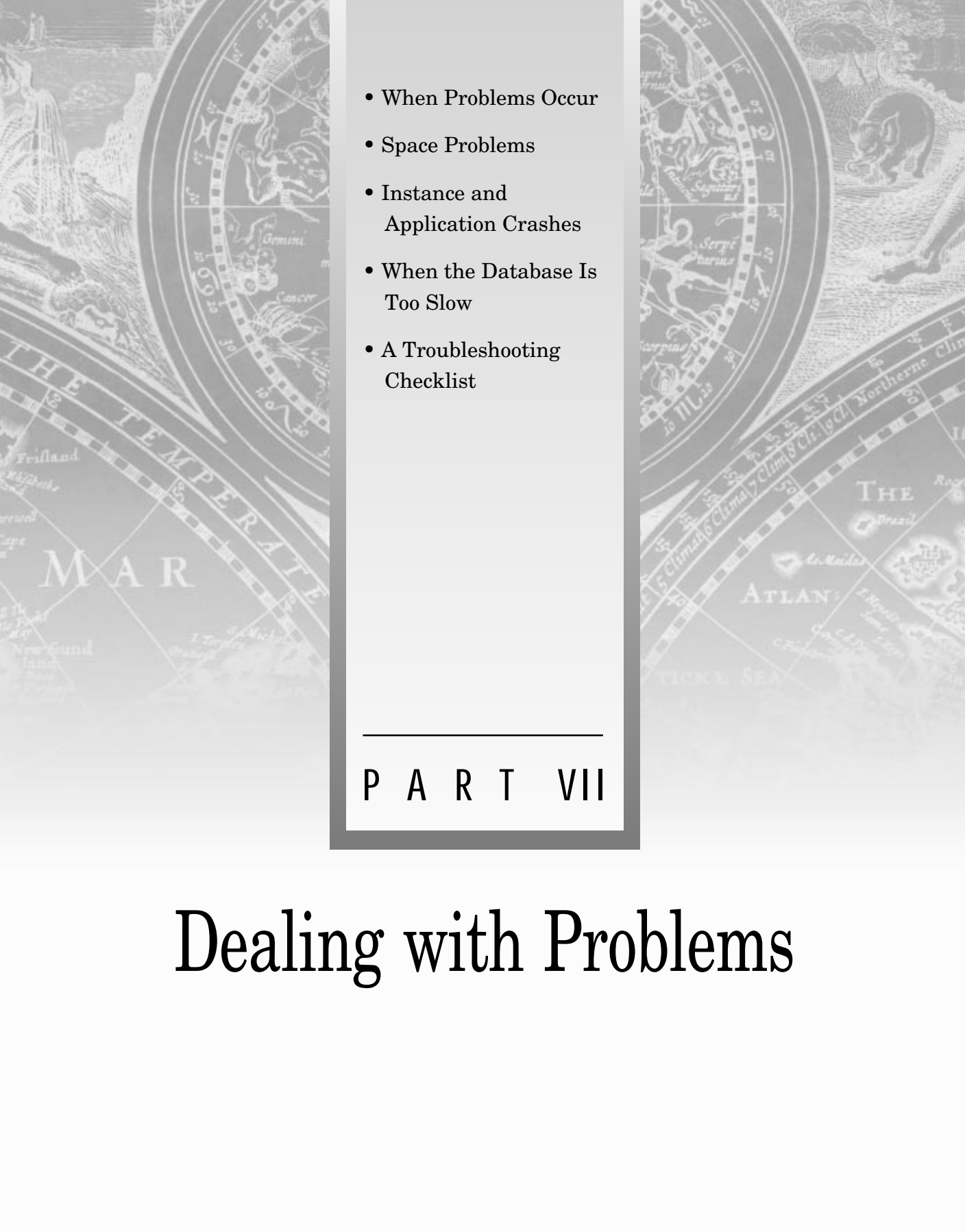
- ◆ Present your projections of need in an easy-to-digest manner. Try to highlight the projects or factors that contribute to key growth items.
- ◆ Finally, give management options. Don't let them shoot down your planning effort and muddy the waters over one little factor. Let them decide if project xyz is to happen or not. If it doesn't, show them how it decreases your disk needs by only 200M and let the rest of your plan stand.

SUMMARY

This chapter presented the highlights of future planning for your databases. It cannot be a complete reference. It also cannot anticipate the local requirements of your management. However, it should serve as food for thought as to how the data collected in the monitoring program can be used to help you prepare for the future. Again, it is not an exact science, but I find it is better to have an imperfect plan than none at all.

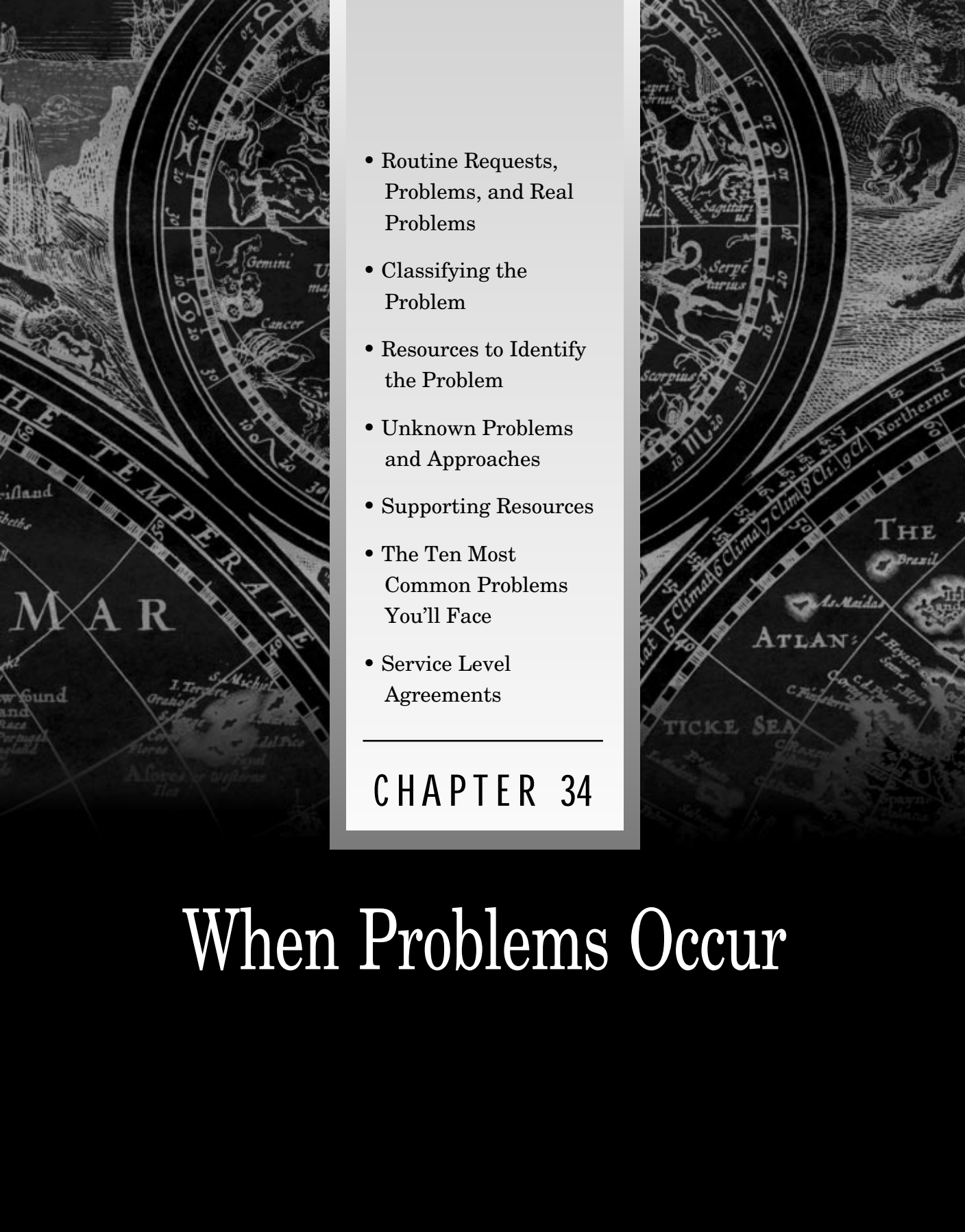
This also concludes the part of this book devoted to the health of your database. It started out with a discussion of what makes up a healthy database. Next, you learned auditing and tuning programs that you can use to assess the health of your instances. In the tuning section, you explored some ways that you can improve the health of your database. This chapter took the proactive health management concepts one step further and presented some long-range planning concepts.

I wish that I could say that if you follow this proactive program you will never have any problems. The realist in me knows that even the best proactive programs can only minimize problems, not eliminate them. That is the purpose for the next several chapters, which discuss the types of problems that you will come across as a DBA and some methods that you can use to solve them.

- 
- When Problems Occur
 - Space Problems
 - Instance and Application Crashes
 - When the Database Is Too Slow
 - A Troubleshooting Checklist

P A R T VII

Dealing with Problems

- 
- Routine Requests, Problems, and Real Problems
 - Classifying the Problem
 - Resources to Identify the Problem
 - Unknown Problems and Approaches
 - Supporting Resources
 - The Ten Most Common Problems You'll Face
 - Service Level Agreements

CHAPTER 34

When Problems Occur

Suppose you spent days planning and configuring your instance. You routinely monitor and graph everything and can predict to within 10 minutes when your tablespaces are going to fill up, but never let that happen. You have a tight security system that prevents any unauthorized actions. Your developers are highly trained and never make mistakes. Unfortunately, in spite of all of your care, preparation and training, you had better be ready for the problems that are about to arise.

Many of these problems are unavoidable. With a system that is as complex as Oracle, you will find little glitches in the software. Perhaps you are an especially large or busy database for your given host platform and find bugs that no one else could possibly discover. Finally, you may have developers who need access to system privileges such as `CREATE TABLE`, and therefore they have the power to cause some nasty little problems.

Note

Some problems cannot be avoided. How well you handle these problems determines whether you're the villain or the hero.

Because problems cannot be avoided, it is important to have an understanding of what to do and have resources in place to help you solve them. This part of the book is designed to provide you with an overview of the problems that typically occur within Oracle and some techniques for dealing with them. It obviously cannot include the details of how to solve every problem, which depends on your system and the version of the Oracle software that you are using. In spite of this, there are some overall techniques that apply across the board and will help you get started.

The other important part of dealing with problems is having the resources available to help you when the problems go beyond the normal realm of annoyances. You probably already have some of these resources. The Oracle documentation, especially that on errors and messages, is a good place to start. Your Server Administrator Guide and concepts manual are also good resources. If you have an Oracle support contract (with Oracle or some other contractor), you can access these resources to solve your problems. Whatever support structure you have, it is important to have everything lined up (telephone numbers, customer service contract numbers, and so forth) before the problems occur and you are plunged into the heat of battle.

This chapter begins with a general overview of problems and problem solving methodologies, then covers some of the resources that can help you classify and resolve problems. Included is a list of most commonly found problems to focus your attention when reviewing later chapters. Finally, you explore the concept of service level agreements, both from vendors with your organization and for your organization to the users that you support.

The remaining chapters of this section discuss the various types of problems that are typically encountered and some of the things that you need to look for and do to solve these problems. Again, you should read and be familiar with where you go to troubleshoot the problem and how you are going to approach it long before it actually occurs. These chapters stress tips to ensure that you do not react too quickly and therefore lose some of that precious data trusted to your database. This part of the book concludes with a checklist that consolidates a lot of information into a relatively simple form. Keeping things simple in a crisis is usually the best approach.

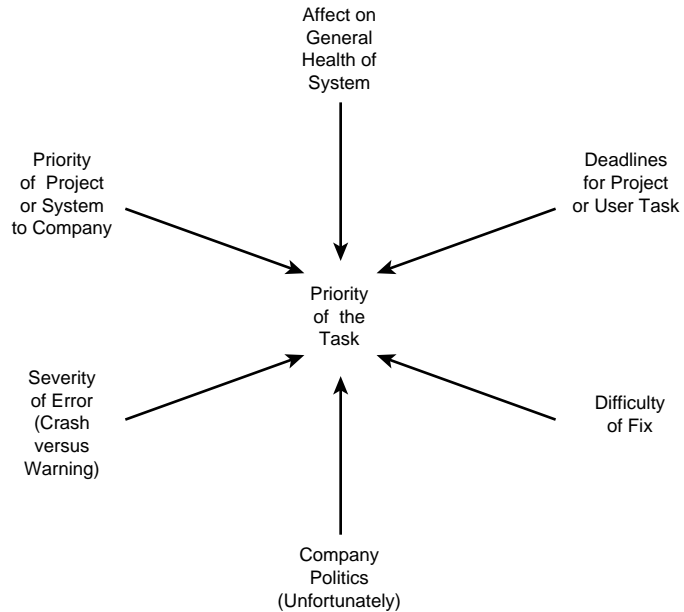
ROUTINE REQUESTS, PROBLEMS, AND REAL PROBLEMS

There are a lot of different users out there. Some might be able to calmly take it when the hard disks on their PCs fail and they have not ever made backups. Others will scream bloody murder if the color of one of the fields on a screen is not the way they want it to be. Your challenge, as a service provider, is to keep a level head and sort out the true priority of problems as they are brought to you. This is not at all easy. It may be harmful to your career and your managers' careers if you try to resolve a production problem before answering a routine information request from a vice president in your company. There are a large number of factors that go into prioritizing requests for your time (see Figure 34.1), but if you do not at least try to sort things out, you are likely to work on the wrong problems and get in trouble for it. The goal of this section is to present some of the things that you have to consider when prioritizing an issue and to enable you to classify tasks into three basic categories:

- ◆ Routine requests
- ◆ Problems
- ◆ *Real* problems

Perhaps the most important consideration when reviewing a problem is to assess the potential impact of the problem on the health of the system. Perhaps you have a tablespace that was too full for a particular developer to perform an upgrade of the software. In itself, this is probably not a critical problem, but what if that tablespace were heavily used by production users? In this case, your production system might come crashing down when it becomes too full to hold end user data. You need to look beyond the immediate problem to the potential affects on the system when defining priorities.

Figure 34.1.
Factors that affect the
priority of a task.



Another important factor to consider is the priority of the system or project to your organization. A problem with a mission critical system normally overrides the complete failure of a development system. There is no absolute scale. You need to use judgment and experience to make these calls. Of course, there is no harm in getting some input from management to be sure.

Project or system deadlines also need to be factored into your decision. A problem with a financial system during the annual closing of the books may override everything else. Perhaps there is not much of a chance that the problem will cause any permanent damage or it may not even repeat itself. People who are under pressure from deadlines are usually not interested in statistics or overall impacts. They just want you to make the system work correctly.

You may be surprised that the previous factors were introduced first, before mention of the severity of the problem. This was not accidental. The severity of the error is usually the first thing that you see when the error message flashes across the screen. I wanted you to think about some of the other factors before you jumped in to solving the most severe problems first (for example, a crash of the test instance before a performance problem caused by excessive table locks in production). Some messages may sound as though the problem is more severe than it actually is (for example, Maximum number of DML locks is exceeded). Conversely, some error messages have a number of potential causes—some are not severe and others are. Therefore, you will usually have to do some investigation before you can accurately determine the severity.

Finally, you should consider the difficulty of the fix when determining what to do. Most successful time management authors tell you to determine the priorities and work from the top of the list. I generally ascribe to this theory; however, I occasionally quickly kill users who have a hung Oracle process so that they can get on about their work. The key is not to let the easy tasks interfere with the big ones. On the other side of the coin, you may find certain tasks are extremely difficult or taxing of system resources. You may have to defer solutions to important problems to periods of lower system use. For example, it's a good idea to perform major tasks, such as tablespace reorganization, after normal working hours when there are fewer interruptions and the system is more lightly loaded (you do not want your concentration broken). Another task that you may find you have to perform after hours is adjusting initialization parameters, which requires you to take down and restart the instance.

It can be quite a challenge to balance all these factors in real life. It's difficult to guess the situations that you may run across or the issues that are local to your organization; however, the following is a list of typical DBA tasks:

- ◆ Design some tables for a new application module that will be built in a couple of weeks.
- ◆ Extend the size of a tablespace that is filling up rapidly.
- ◆ Document the configuration of a new instance that was installed.
- ◆ Investigate why some SQL*Net connections are dropping out every now and then.
- ◆ Rebuild a test instance that was altered for some stress testing.
- ◆ Add an account for a new vice president.

Here are some solutions and the order in which you should attack:

1. Start with the account for the vice president. Vice presidents are typically sensitive and might complain to your management if they have to wait. It's also a good idea to set up regular user accounts promptly (usually within the same day if there are not any major evolutions or problems with the system). This usually promotes a favorable first impression and is a quick task that really will not crimp your schedule on most days.
2. Extend the size of the tablespace that is filling up. It can be really ugly when a tablespace runs out of space (applications crash, people get upset, and so on). It's a good idea to head off problems whenever possible.
3. Investigate SQL*Net connections that are dropping out. Again, you're giving high priority to user problems. It may be that there is nothing you can do about these problems (perhaps they are with the user's local area network), but the DBA is often the one who has to prove that it is someone else's problem.

4. Next, work on the document for the configuration of the new instance. (It's just a personal preference, but I like to get a task completely done and out of the way.) Don't make documentation the lowest priority. This leads to operational systems that lack documentation.
5. Design the new tables. Although this can often be the most enjoyable task (you get to design a system that will make everyone's life easier), it is not due for a while. It should move up on the priority list as the due date approaches.
6. Rebuild the test instance. It wasn't specified, so it's assumed that this instance is not needed for some time, and this is a pretty routine task.

Enough food for thought on this subject. There are probably other factors that you should figure in that are important in your local environment. The relative weighting of the various factors is also usually a local function (there are places where people could die if the computer systems stop working). Just try to think about the various factors and try to classify the problems as routine requests, problems, or real problems (A-B-C, or whatever works for you).

CLASSIFYING THE PROBLEM

Before you can decide on the importance of the problem, you need to classify it. This is not to be confused with the classification of the priority of the project. Instead, this is a purely technical evaluation of what is causing the problem. Sometimes you can nail the problem down right away (tablespace SALES is completely full and you need to add additional data files). At other times, you may be able to determine only the rough cause of the problem and then rely on resources such as Oracle support to figure out the details of the problem.

The first step you need to take is collecting all the information available related to the problem. The best thing that you can have is a print of the screen that showed the error message. If your environment supports this (and most PC-based environments do), teach people how to use it. One of the greatest problems you will face is someone saying that they had an error message that said something about a table. I have often had to spend a lot of time trying to reproduce an error just to get a good error message with which I could work. This may be impossible with intermittent problems, so have people give you the exact error message whenever possible.

There is often a wealth of information beyond screen messages. The first sources, especially in batch applications, are log files that the application makes. The Oracle error messages and error codes can be read from this file. Hopefully, your developers also include some basic processing status in larger applications so that you can tell where the problems occurred (for example, beginning download or running an import). This will help you figure out which statement was being executed when the problem arose.

A good source of information for serious system problems are the Oracle log and trace files. The alert log can also record major system events that are occurring (such as log file switches), which can show you what else was happening at the time of the problem. Your goal is to be able to look at the date and time of the trace and log files (`ls -l` or `ls -lt` under UNIX) to determine which, if any, of them were created or updated around the time of the error. An important part of preparing to deal with problems is mapping out (in your head or notebook) where the log and trace files for your instances and products such as SQL*Net are located so that you do not have to waste time hunting in the middle of a crisis. The following example shows a typical alert log file (this one shows some database administrative activities, not problems with the system):

```
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:20 1995
```

```
Starting up ORACLE RDBMS Version: 7.1.4.1.0.
```

```
System parameters with non-default values:
```

```
processes                = 50
license_max_sessions     = 25
control_files             = %RDBMS71_CONTROL%\ct11.ora, %RDBMS71_ARCHIVE%\ct11.ora
db_block_buffers         = 400
log_archive_dest          = %RDBMS71_ARCHIVE%
log_buffer                = 65596
log_checkpoint_interval   = 1000
row_locking               = ALWAYS
sequence_cache_hash_buckets= 10
max_enabled_roles         = 8
remote_login_passwordfile= EXCLUSIVE
distributed_lock_timeout  = 10
distributed_recovery_connection_hold_time= 0
mts_servers               = 0
mts_max_servers           = 0
mts_max_dispatchers       = 0
open_links                = 20
audit_trail               = NONE
sort_area_size            = 262144
sort_area_retained_size   = 262144
db_name                   = oracle
background_dump_dest      = %RDBMS71%\trace
user_dump_dest            = %RDBMS71%\trace
max_dump_file_size        = 5120
```

```
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:22 1995
```

```
PMON started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:23 1995
```

```
DBWR started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:23 1995
```

```
LGWR started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:27 1995

ALTER DATABASE oracle MOUNT
Sat Feb 18 15:12:29 1995

Completed: ALTER DATABASE oracle MOUNT
Sat Feb 18 15:12:29 1995

ALTER DATABASE NOARCHIVELOG

Completed: ALTER DATABASE NOARCHIVELOG
Sat Feb 18 15:12:29 1995

ALTER DATABASE oracle OPEN
Sat Feb 18 15:12:32 1995

Thread 1 opened at log sequence 1

Current log# 2 seq# 1 mem# 0: C:\ORAWIN\DBS\wdblog2.ora
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:12:32 1995

SMON: enabling cache recovery
Sat Feb 18 15:12:42 1995

SMON: enabling tx recovery
Sat Feb 18 15:12:43 1995

Completed: ALTER DATABASE oracle OPEN
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Sat Feb 18 15:18:08 1995

Shutting down instance (immediate)

License high water mark = 1
Sat Feb 18 15:18:09 1995

ALTER DATABASE oracle CLOSE NORMAL
Sat Feb 18 15:18:09 1995

SMON: disabling tx recovery

SMON: disabling cache recovery
Sat Feb 18 15:18:11 1995

Thread 1 closed at log sequence 1

Current log# 2 seq# 1 mem# 0: C:\ORAWIN\DBS\wdblog2.ora
Sat Feb 18 15:18:11 1995

Completed: ALTER DATABASE oracle CLOSE NORMAL
Sat Feb 18 15:18:11 1995

ALTER DATABASE oracle DISMOUNT

Completed: ALTER DATABASE oracle DISMOUNT
```


It's a good idea to print out the applicable log files or screen prints and keep them handy in case what you thought was a minor problem turns out to be more serious and you need data to discuss with Oracle Tech Support or other support groups. Your goal now that you have the data is to look at this information and reference books, such as the messages and codes manual, to determine what the problem is. You may find that you are uncertain as to the exact problem based on this information. You may choose to issue some queries to the database to determine the size of a given table, the number of extents, or the amount of free space in a given tablespace. You may choose to run the tuning or utilization reports to get access to a range of supporting information about your database or instance.

Now that you have the information, perhaps some gained by issuing follow-up queries, it is time to classify your problem. First, you have to decide where the problem lies—for example, configuration parameters, rollback segments, or so forth. You also have to make the call as to whether this is a routine task, a problem, or a real problem. Depending on the problem, the answer may be as close as your messages and codes manual (it tells you to increase this or remove that). However, you may be stuck and need to call in support. (There is more information about support resources in later sections.)

There are a large number of potential problems, most of which you might never see. For purposes of this book, I have divided them into the following areas:

- ◆ *Space problems.* Especially in systems that process large volumes of information, you are continuously running out of space. Hopefully your monitoring program will catch this, but users can occasionally surprise you (more about these problems in Chapter 35).
- ◆ *Oracle crashes.* These can be the most severe errors because the whole Oracle system goes down, and it often points to a bug in the software that you cannot fix yourself (more about these problems in Chapter 36).
- ◆ *Speed problems.* “The system is too slow” is a continuous complaint, but sometimes the problems are real. (You learn how to separate true problems and solve them in Chapter 37.)
- ◆ *Hardware or operating system problems.* These normally are not within the DBA's area to solve. However, these problems can do damage to your data files. The chapter on backup and recovery (Chapter 14) may come in handy if you run across these types of problems.

RESOURCES TO IDENTIFY THE PROBLEMS

Often it is more important to know where to find the answers than to know the answers themselves. This is very true with Oracle in that the system is just so big that very few could memorize all the possible problems and solutions. Combined

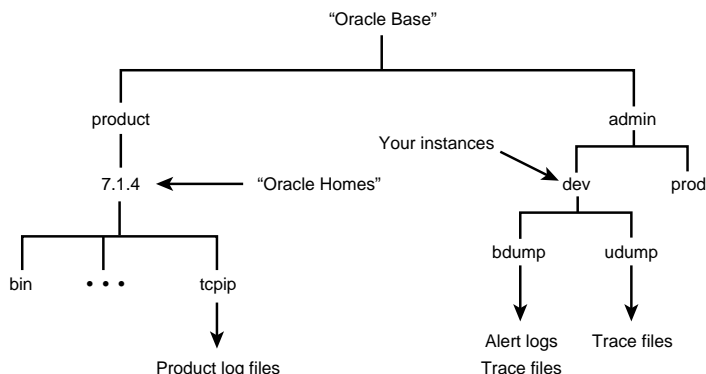
with the fact that new problems are found every day at some location around the world using Oracle, it is important to know where to get help when the nonroutine Oracle problems creep up in your world. This is not to say that you will know the answers to many of the problems that arise. When you run low on space for the tenth time this month, you will probably know the `alter tablespace add datafile` command by heart. This section deals with the resources to solve the problems that separate the really hot DBAs from the rest of the crowd.

Tip

It often is more important to know where to find the answers than to know the answers themselves.

You have already looked at the set of resources that will help you solve the vast majority of problems that you will run across. As a quick refresher, the location of most of the common log and trace files for systems using the OFA architecture is presented in Figure 34.2. For those of you who have never had the pleasure of reviewing an Oracle system trace file, they are really quite simple. They contain the typical header with the Oracle copyright notice and the name of the Oracle background process that detected the problem. After this, they list a series of Oracle error messages and warnings, just as you would see on your screen if the problem occurred with one of your SQL scripts. The key to these files is that they are the only output mechanisms for the background processes. The nice thing about this setup is that the files are written with the name of the process and then a sequential number. These files should stay around until you delete them; therefore, you have them for future reference.

Figure 34.2.
Location of common log
and trace files.



The next set of resources is the Oracle documentation set. You need to understand how Oracle divides its documentation. The messages and codes manual contains a listing of the error messages that are common to all releases of the product. I usually start here to look up an error message. The various Oracle products, such as SQL*Net, have their unique error messages in their own documents, usually located in an appendix near the back. Finally, because Oracle runs on a wide range of operating systems, there are operating system-specific error messages that are located in the reference manual that Oracle provides for your operating system. (It usually has the purple stripe and is the one of the very few that will specifically have your operating system name in its title.) Between these references, you can usually find all the error messages that you receive and a hint as to what to do about the problems. Sometimes the solution is something that you do not want to hear (for example, call Oracle support, which usually means a big problem, especially for those locations that do not have Oracle support contracts).

Finally, there are always your fellow Oracle DBAs as resources. It can be very useful to know a couple of colleagues who have similar systems and are willing to exchange information. True, you may get a call when you are having a busy day from one of these folks who wants some help on a problem, but they can also bail you out when disaster strikes you. It is especially useful if you build up a network of people who have the same set of problems that you do—large data warehouses, your particular host computer operating system, and so forth. Oracle support typically will want to start with a list of error messages and something with which they can work. Your fellow DBAs may be a little more patient at taking you through the troubleshooting process or may be able to give you a very quick answer (because they had the same problem last week).

UNKNOWN PROBLEMS AND APPROACHES

You looked at every log and trace file on the system. You ran tuning and utilization reports twice. However, you still do not have a clue as to what the problem was. Perhaps the user forgot the exact message on the screen or it was a batch application that did not log the message. Perhaps all you know is that your instance crashed without a hint as to why it happened. Now what do you do?

To be honest, I hope you never run into this situation. It can be very frustrating and even dangerous to your system—yes, dangerous. You may have to let the system continue to operate until you see the error occur again, and the next time you might lose some important data. Worse still, you may have to try to re-create the error and then it will be your fault if data is lost. Therefore, it is always best to proceed cautiously when you run across these situations.

The first question that you should ask yourself when you run across a completely unknown problem is what the risk to your system from this problem will be. If a user is bombing out when running a report, you are probably fairly safe. You may have annoyed users, but it is unlikely that you will lose data. If a major database update procedure is crashing or the whole instance is going down, always ask yourself two questions:

When was my last backup and am I protected with archive logging?

What would be the harm of suspending this process until a fix or work-around is found?

These are the types of issues wherein a support contract can come in handy. Support people probably cannot provide much help if all you have is a user who forgot what the screen message was. However, if you have a particular type of crash, they often can provide some guidance once you describe the situation. More importantly, they can provide operating system-specific guidance as to how to increase the chances that the next time this problem occurs, you will get some data that can help find the problem.

The final recommendation in this type of situation is to keep users and your management informed. Sending out broadcast messages (via e-mail or phone mail) to let all interested parties know what is going on is usually helpful. At least they know not to throw away their data sheets after entering data. It is also helpful to keep management people informed and let them concur with your actions. That way, they will not be surprised when things happen. If they know about issues beforehand, they feel as though they are a part of the process and are a little less nervous.

SUPPORTING RESOURCES

In many cases, the Oracle documentation contains enough information to tell you exactly what you need to do. In other cases, the answer may be relatively straightforward (for example, your tablespace is full) and you can just proceed without any assistance. However, for those other times, it is nice to know where to call before you are in desperate need. I keep a sheet with key phone numbers, network addresses, and other resources in a binder on my desk to help find what I need quickly.

The first supporting resource has been mentioned earlier. That is a group of fellow DBAs whom you can call on in an emergency. Typically, you should not expect these folks to drop whatever they are doing and come over to help. However, they are usually willing to tell you whether they had a similar problem and how they solved it. They can also be useful if they have applied a patch that you find you might need (faster than it could be shipped from Oracle).

A second resource is Oracle support. It provides 24-hour-per-day coverage, although this type of maintenance contract is more expensive than the normal working-hours type of support. Oracle support has access to a database that contains pretty much every problem that has been reported to Oracle. There are also solutions for most of these problems, although a few are software bugs that are deferred until a future release of the Oracle software. If you have a contract with Oracle, you should have the number for Oracle support and your CSI number readily available. The CSI number identifies you as a customer. It is specific to a particular host computer at your organization, although the CSI numbers for PC-based products are usually grouped by all PCs at the site.

Next in line to provide support are some unique resources that have recently become available to many of us—those that reside on the Internet. The Oracle Newsgroup (`comp.databases.oracle`) has an incredibly high volume of traffic (80–100 postings per day). There is usually someone who is willing to take a shot at answering your problems, especially if your problem is challenging. It may take several days to get an answer, but there are some really experienced DBAs who monitor this forum. The Internet mail system can also be useful. Rather than wait for a problem report or patch to be shipped via regular mail, the folks at Oracle support can send literature and relatively small patches via mail. I have had a patch from Oracle arrive within an hour of when I asked for it (even Federal Express cannot beat that kind of service). Finally, there are a number of Oracle resources available through ftp (file transfer protocol) and World Wide Web sites. The World Wide Web is difficult to keep up with because, as of this writing, the number of sites on the Web was doubling every 53 days. However, when you have some free time, it might be useful to browse a few of these places so that you are familiar with what is out there that might be helpful when problems come up:

- ◆ <http://www.oracle.com> (http indicates a World Wide Web site)
- ◆ <http://www.ioug.org>
- ◆ <http://www.bf.rmit.edu.au/Oracle>
- ◆ <http://www.lpac.ac.uk/SEL-HPC/Articles/DBArchive.html>
- ◆ <ftp://ftp.bf.rmit.edu.au/pub/Oracle/>
- ◆ <ftp://ftp.Informatik.Uni-Oldenburg.DE/pub/oracle>

THE TEN MOST COMMON PROBLEMS YOU'LL FACE

Ten seems like a reasonable number to deal with when discussing the problems most frequently faced by DBAs. Most of the problems that come up in an Oracle instance

fit into these categories. Your job is to become familiar with how to handle these situations and be ready to deal with them when they come up:

1. A disk drive or the operating system fails and you have to recover your Oracle instances and/or data files.
2. Developers accidentally delete a table that they really need.
3. The directory where you store your archive log files or alert log fills up.
4. A user forgets the password and you need to reset it.
5. A user lacks permissions to perform a particular activity.
6. A user locks up one of the Oracle database jobs and needs you to kill the session.
7. One of your rollback segments runs out of space.
8. One of your tablespaces is full.
9. One of your tables has reached its maximum number of extents.
10. A user feels that the database is too slow.

The last one is the most common problem with Oracle databases. Most of these topics are addressed in the upcoming chapters. The rest are covered under the routine maintenance discussions in other chapters. Killing sessions is a task that you should be able to find in the menus in SQL*DBA (try the monitor session option to figure out the session ID, and then choose the kill session option from the menu). Take these items as food for thought so that you can be ready when they arise.

SERVICE LEVEL AGREEMENTS

One of the first things that you get hit with from hardware and software vendors when discussing support contracts is service levels. You have to figure out the balance between what you are willing to pay and how much support you need. Typical variables in these service level agreements are response time, services provided, and hours of availability. You need to consider the hours of support carefully when setting up Oracle service agreements. If you have an important production instance in which you, the DBA, will be called in during the middle of the night, it is reasonable to expect that you will be able to call Oracle support during this time. Many of the serious problems require some support from Oracle. Your job is to make your management understand this situation.

Let's take this one step further. If vendors can use the terms of the service level agreement against you, you should also have an understanding, up front, with your users as to the service levels that you will provide. Users tend to live in their little worlds and not see all the other demands on your time. All they see is your support when they have a problem. You should be supportive and help these people get their

jobs done. However, there needs to be some balance when it comes to demands on your time, and this balance should be agreed to by your management in advance. For example, in most places you should not get called at home when a user has a routine question that could wait until the next business day. In almost every case, companies expect support staff to come in when a production system is down, but what about when development instances lock up or run out of space? That is an issue that you should negotiate in advance to minimize the disruptions and demands on your time. Who knows—perhaps you can even get some time off work to do other things with your life.

SUMMARY

This chapter presented the problems that can crop up in an Oracle database from a high-level perspective. You learned the basics of determining the severity of a problem. The weighting of factors will vary between different locations. Deadlines are very important in development shops; other shops favor production instances over everything else. This chapter presented tips on classifying the problem and trying to figure out what to do about it. Several resources were given that you might call on to help you solve the problem. Finally, you explored the concept of service level agreements. You should understand the terms of your agreements with support vendors and ensure that you have the level of support that you need. You should also have an understanding with your users as to what levels of service you will be providing to them—and when.

The next several chapters present some of the specific problems that may arise. Chapter 35 deals with typically the most frequent problem—space problems in the tablespaces. Chapter 36 discusses typically the most severe problem—crashes of your Oracle instance background or user processes. Chapter 37 addresses speed issues, and Chapter 38 presents a troubleshooting checklist.

Remember to prepare for problems in advance. Have the support numbers (telephone numbers, CSI numbers, and so forth) ready in advance. Keep all the information that you need (maps of disk usage, copies of configuration, tuning, utilization, and security reports, and so forth) handy—in your desk or on the wall. Whatever form this takes, take time up front so that when you are in the heat of battle, you are ready to move as quickly as possible. Companies have become extremely dependent on their computer systems. Every minute counts to users who are waiting for the system to return. Your up-front preparation can make the difference between your being a hero or a villain when a problem occurs.

- 
- Identifying the True Problem
 - Cleaning Out Tablespaces
 - Expanding Tablespaces
 - Compressing the Number of Extents
 - Alternatives:
Reducing Data Storage
 - Keeping the Data Definition Language (DDL)
-

CHAPTER 35

Space Problems

When discussing problems, I rank space problems first because they are the ones that I most frequently run across. They tend to crop up in any database wherein you routinely add data to the system. One of the goals of your monitoring program is to keep track of space utilization and correct problems before they occur. However, there are any number of reasons why even in the best monitored database, you will still see problems. You may have special cases, such as when you bring a new application on line. The developers provided size estimates, but they turned out to be too low, so you run out of space. Other factors, such as unusually large business volume, may catch you by surprise and fill up your tablespaces more quickly than you anticipated.

Whatever the cause, you need to be able to deal with space problems on your system. This chapter begins with an important topic—identifying the true problem. In many cases, Oracle runs into a problem that can have many causes. It is your job to figure out which of these causes caused your problem and then take the appropriate corrective action. Next, you explore some of the more common actions that relate to space problems—cleaning out tablespaces, expanding tablespaces, compressing the number of extents, and reducing data storage. Finally, you learn to keep the data definition language for objects in your database.

Warning

Most space problems are “hard” problems. You cannot work around them and you cannot live with intermittent space problems.

One final general note about space problems: Most of these problems are “hard” problems. When you run out of space in a tablespace, it prevents you from adding any data to the table that ran out of space. You cannot work around it or live with intermittent problems. Therefore, you should be ready to take action quickly on these problems, especially for production databases.

IDENTIFYING THE TRUE PROBLEM

To identify the true cause of a space problem, it is helpful to review the basics of Oracle storage. Figure 35.1 provides a quick overview of the basic concepts. The key to understanding space problems is the concept of extents. When Oracle adds space for a given database object (for example, a table), it demands a number of contiguous blocks. The size of the extent is controlled by the storage parameters that you set up for the table (or inherit by default from the tablespace or database). It does not matter how much total space you have available; it needs to be contiguous for Oracle to enable you to create the extent. Therefore, if you fail to allocate an extent, it could

be due to a lack of total space in the tablespace or the lack of contiguous blocks for that next extent. Here is an example:

```
SQL> list
1 create table another_test
2 (first_column char(10),
3 second_column number(4))
4 tablespace user_data
5 storage (initial 50M
6          next 2M
7          pctincrease 0
8          minextents 1
9*         maxextents 100)
SQL> run
1 create table another_test
2 (first_column char(10),
3 second_column number(4))
4 tablespace user_data
5 storage (initial 50M
6          next 2M
7          pctincrease 0
8          minextents 1
9*         maxextents 100)
create table another_test
*
ERROR at line 1:
ORA-01652: unable to extend temp segment by 25600 in tablespace USER_DATA
```

Figure 35.1.
The basics of Oracle storage.

Datafile							
Table 1 Extent 1	Table 2 Extent 1	Table 1 Extent 2	Free Extent	Table 4 Extent 1	Free Extent	Table 3 Extent 1	Free Extent

You can tell which is your problem when you fail to allocate extents by running the utilization report (see Chapter 30). In this report, you have two key queries. The first tells you the total amount of free space in each of your tablespaces. The other section is the list of free extents in the tablespaces. Look at the tablespace in question and find the largest available extent (it will be at the top of the list for that tablespace).

```
#####

Tablespace Utilization

Table_Space      Free_Bytes      Total_Bytes    %Used
-----
ROLLBACK_DATA    2,734,080      3,145,728     13.0
SYSTEM           3,192,832     10,485,760    69.5
TEMPORARY_DATA   2,095,104      2,097,152
USER_DATA        3,065,856      3,145,728     2.5

#####
```

#####

Tablespace Free Segments Report

Table Space	Bytes
-----	-----
ROLLBACK_DATA	2,734,080
SYSTEM	3,192,832
TEMPORARY_DATA	2,095,104
USER_DATA	14,336
USER_DATA	3,051,520

#####

To find the size of the extent that the table wanted to allocate, you can usually look at your error message and find the size of the extent that it failed to allocate (in blocks, *not* bytes). If you do not have this data handy, you can issue a query similar to the following example:

```
SQL> select * from dba_tables where table_name = 'CUSTOMERS';
```

OWNER	TABLE_NAME	TABLESPACE_NAME
-----	-----	-----
CLUSTER_NAME	PCT_FREE	PCT_USED
INITIAL_EXTENT	NEXT_EXTENT	INI_TRANS
MAX_TRANS		
-----	-----	-----
MIN_EXTENTS	MAX_EXTENTS	PCT_INCREASE
B	NUM_ROWS	BLOCKS
EMPTY_BLOCKS	AVG_SPACE	
-----	-----	-----
AVG_ROW_LEN	DEGREE	INSTANCES
CACHE		
-----	-----	-----
JGREENE	CUSTOMERS	USER_DATA
	10	40
		1
		255
4096	4096	
1	121	10
N		0
0		0
0	1	1
N		

```
SQL>
```

If there is enough space in the tablespace, but no free extents that are large enough, you get into compressing extents in the tablespace (de-fragmentation). This gives you a single large free extent at the end of the tablespace that can hold the extent that you want. If there is not enough space in the tablespace, you have to either clean out some data to make space or add additional data files to the tablespace. Be careful if you clean out some tables to make room to check fragmentation. You may create a large number of small extents when you remove tables. This may not create a free extent large enough to hold your new data, and therefore you may have to de-fragment the tablespace after you finish cleaning it out.

Another cause for failing to allocate space in a tablespace is a lack of permissions. Although it is true that the `create table` command is checked to see whether the initial extent is allocated, tablespace quotas can prevent users from adding data to these tables. Every time you add an extent to a table with quotas in place, Oracle

checks to see whether the total amount of space used by the owner of the table exceeds the quota assigned by the DBA. Therefore, a user may fail to allocate an extent because the owner of the table is over the limit. It is important that you keep track of the quotas you assign and check for this condition if you have a problem allocating space in a tablespace that has plenty of contiguous blocks available. Remember, once quotas are activated for an instance, everyone needs to have an assigned quota to own database objects (this can be a real pain). Please reference the following example:

```
SQL> get q_check
 1 select q.username, q.tablespace_name, q.bytes Quota, sum(e.bytes) Used
 2 from dba_ts_quotas q, dba_extents e
 3 where q.username = e.owner
 4 and q.tablespace_name = e.tablespace_name
 5* group by q.username, q.tablespace_name, q.bytes
SQL> /
```

USERNAME	TABLESPACE_NAME	QUOTA	USED
JGREENE	SYSTEM	759,808	759,808
JGREENE	USER_DATA	40,960	40,960

SQL>

Finally, you need to take special care for certain types of objects. The most important resource that you have is the system tablespace. It contains all the tables that make up the Oracle data dictionary and is essential for your instance. (You can take any tablespace that you want out of the system except the system tablespace.) It is also the location that stores some special objects such as stored procedures. Therefore, if you have a lot of stored procedures you should keep a close eye on the system tablespace. Other especially sensitive areas to keep an eye on are the rollback segments. These are somewhat confusing beasts in that they function in the background. They are usually created when you create your instance, and you never have any direct interaction with them. They have some special storage considerations that you learn more about in Chapter 42. For now, remember that the storage of rollback segments is different from that of tables and, much like archive log files, they can cause some nasty problems.

CLEANING OUT TABLESPACES

Let's say that you have run out of space in a given tablespace and there is no free disk space for you in which to add data files to your tablespace. Perhaps you have run out of space, but know that the tablespace is loaded with junk that should be cleaned out. The pack rat in you fears throwing anything away. It is sure that whatever you remove will be needed soon after you get rid of it. These are real problems that a DBA who has a space limitation faces.

Cleaning out tablespaces generally involves dropping tables that are no longer needed. Although the `drop table` command is relatively simple to figure out, finding

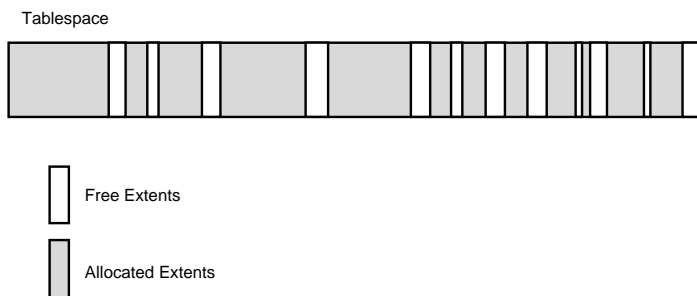
the right tables to get rid of is often not so simple. This is especially true when you have developers who are also pack rats and do not keep track of what they create. These types are often more open to the idea of removing the tables and indexes if you offer to make a special export of these tables and store the tape in case the data is needed in the future. Once this is done, you can drop the tables. Remember that as the DBA, you can drop any user table that you want by using the `drop table` command. If the table is referenced by a foreign key in another table, you have to add the words `CASCADE CONSTRAINTS` at the end of the `drop table` command to drop these foreign keys. The following is the basic command format:

```
drop table owner.table_name [CASCADE CONSTRAINTS];
```

In a production environment, you typically have disks to support your needs. If you run low on disks, you then need to tell management that you either have to get rid of data or get more disks. Development environments tend to be somewhat different. The developers are often within the same information systems organization that you are. You are usually put in an awkward position when it comes to fighting with them as to whether you really need to have scratch disks for DBA purposes or whether these disks should be allocated to the developers (usually with the excuse of “just until next fiscal year when we’ll pay you back—trust us”). In these situations, I usually try to head off problems before they happen. If you can get management to buy into giving the developers control of their own tablespaces and set a limit on their disk space, give it a try. You can then put the developers on the hot seat to figure out how to manage their space. It is amazing how they can find things that are no longer needed when their manager asks them, but they cannot find so much as a single byte when you need it.

One final point on this theme. If you delete a number of tables, you may get a large amount of space back in your tablespace, but you may not get many large, contiguous extents in which you can add new extents. This is especially true if you are cleaning out test tables, which typically have small extent sizes, and replacing them with extents from production tables. You may wind up with a highly fragmented tablespace that looks like a hunk of Swiss cheese (see Figure 35.2). The solution to this is to de-fragment the tablespace (as discussed a little later in this chapter).

Figure 35.2.
Highly fragmented
tablespaces.



EXPANDING TABLESPACES

If you cannot delete any of the existing data, you have the choice of forbidding the addition of any data to your database—unlikely in the environments that I have been in—or expanding your tablespaces. In most production environments, you do not get the option of deleting data or forbidding data addition; rather, you have to add data files. This trend toward more and more data online has been occurring throughout most of the computerized world for decades and is unlikely to stop any time soon. Computer disk drive manufacturers keep lowering the prices and increasing their capacities, so get used to it and become comfortable with the idea of adding data files to your system.

First, you want to deal with the question of where to add the new data files and how large to make them. Let's start with the question of how large to make the data files. If you have been routinely monitoring the size growth of your data on a tablespace-by-tablespace basis, you should have a good feel about how large to make the new data files. You have to go through a bit of a tradeoff balancing the frequency with which you will be expanding the size of the tablespace against the needs of the other tablespaces for data space. If you make the new data file small, you will probably be coming back in a few months and expanding it again. If you expand the size of the tablespace so that it will not need expansion for the next five years, you will probably have trouble finding space for your other tablespaces when they fill up.

The other question that you have to deal with when planning new data files is where to add these files. Remember when you planned a layout of your data files to ensure that input/output loads were balanced across disk drives and controllers? You need to keep that plan in mind when you are planning tablespace expansion. Reviewing the basic storage concepts of Oracle, you have no control over where Oracle puts data within a tablespace. Therefore, you have to assume that Oracle may write to or read from any of the data files in a tablespace. As a quick review to help you find out where your existing data files are, the following command is useful:

```
SQL> select * from dba_data_files;
```

```
FILE_NAME
```

FILE_ID	TABLESPACE_NAME	BYTES	BLOCKS	STATUS
C:\ORAWIN\DBS\wdbrrs.ora				
3	ROLLBACK_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbtemp.ora				
4	TEMPORARY_DATA	2097152	1024	AVAILABLE
C:\ORAWIN\DBS\wdbuser.ora				
2	USER_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbsys.ora				
1	SYSTEM	10485760	5120	AVAILABLE

You may also be curious as to where your redo log files are because they will be competing for input/output bandwidth. The following is a quick command to get this information:

```
SQL> select * from v$logfile;
```

```

      GROUP# STATUS
-----
MEMBER
-----
          2
C:\ORAWIN\DBS\wdblog2.ora

          1
C:\ORAWIN\DBS\wdblog1.ora
```

Enough for the planning aspects. By now you should know where you want to put the new data files and how large to make them. All you need is the format of the command that will help expand the size of your tablespace. Because you are making a modification, you will be using one of the alter commands in Oracle—in this case, the alter tablespace command. Here is the simple format for this command:

```
alter tablespace tablespace_name add datafile new_file_name size new_file_size;
```

The following example adds a new file called c:\orawin\dfs\system2.dbf to the system tablespace with a size of 1M:

```
SQL> select * from dba_data_files;
```

```

FILE_NAME
-----
FILE_ID TABLESPACE_NAME          BYTES    BLOCKS STATUS
-----
C:\ORAWIN\DBS\wdbrrbs.ora
      3 ROLLBACK_DATA              3145728    1536 AVAILABLE
C:\ORAWIN\DBS\wdbtemp.ora
      4 TEMPORARY_DATA             2097152    1024 AVAILABLE
C:\ORAWIN\DBS\wdbuser.ora
      2 USER_DATA                  3145728    1536 AVAILABLE
C:\ORAWIN\DBS\wdbsys.ora
      1 SYSTEM                     10485760   5120 AVAILABLE
```

```
SQL> alter tablespace system
      2 add datafile 'c:\orawin\dfs\system2.dbf' size 1M;
```

Tablespace altered.

```
SQL> select * from dba_data_files;
```

FILE_NAME				
FILE_ID	TABLESPACE_NAME	BYTES	BLOCKS	STATUS
C:\ORAWIN\DBS\wdbrrbs.ora 3	ROLLBACK_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbtemp.ora 4	TEMPORARY_DATA	2097152	1024	AVAILABLE
C:\ORAWIN\DBS\wdbuser.ora 2	USER_DATA	3145728	1536	AVAILABLE
C:\ORAWIN\DBS\wdbsys.ora 1	SYSTEM	10485760	5120	AVAILABLE
C:\orawin\dbs\system2.dbf 5	SYSTEM	1048576	512	AVAILABLE

One final point before leaving this discussion on expanding your tablespaces: Work out in advance with the system administrator the disks and file systems to which you will have access. Keep a list or picture (for smaller instances) of where your data files are on the various disks. When a problem occurs, or your routine monitoring indicates that you will need to add space soon, you can quickly find out where things are located and what your alternatives are. Then ask the system administrator for a specific disk (if a new disk is needed and you have not already been given access to it) or simply proceed to add the data file. There are two reasons for doing things this way. First, you are proactive rather than reactive. (I am very uncomfortable reacting in crisis mode.) Second, although system administrators are usually quite talented in terms of computers and the operating system, they should not be assumed to be familiar with Oracle internal structures and ways of doing business. Most of the decisions as to where to put things should be based on a knowledge of what happens to the various tablespaces during database operations. This is normally within your area of expertise and not that of the system administrator.

COMPRESSING THE NUMBER OF EXTENTS

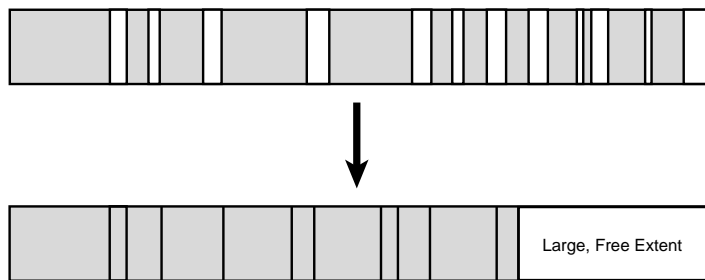
This section is devoted to two different problems that have similar solutions. The first problem occurs when your database objects (tables and indexes, most likely) get near the maximum number of extents that they can reach. When you reach this number, Oracle will stop you from adding data to that object, no matter how many large free extents you have available. Therefore, you need to reduce the number of extents that the object is occupying. This process is called *compression*.

There is another reason why you may wish to consider compressing the extents of a given object, even if you are not approaching your limits. If you have a large number of extents for a given object, Oracle may choose to scatter these extents in multiple locations on a given disk or even across multiple disk drives that store data files for the tablespace. When you issue a query, it may be somewhat slower because

you have to wait for the heads to move around on a disk drive or for queries to be completed to different disk drives. Therefore, it is generally more efficient to have a given object occupying as few extents as possible—preferably one. The exception to this rule is data warehouses wherein a given table or index is the only object in a tablespace. Here, you are guaranteed that extents from other objects will not get between the extents for your table or index, and therefore you can feel free to set your extent size based on your needs. (I used extents that were up to 2G in this case because that was the maximum size of the file system on that operating system.)

The other problem is de-fragmentation, an extension of the concept of compressing extents. This concept can be applied to an entire tablespace to not only compress the extents for the objects in that instance, but to ensure that you have a single large free extent at the end that you can divide as needed. This eliminates the Swiss cheese effect in your tablespaces, as shown in Figure 35.3.

Figure 35.3.
De-fragmentation
of a tablespace.



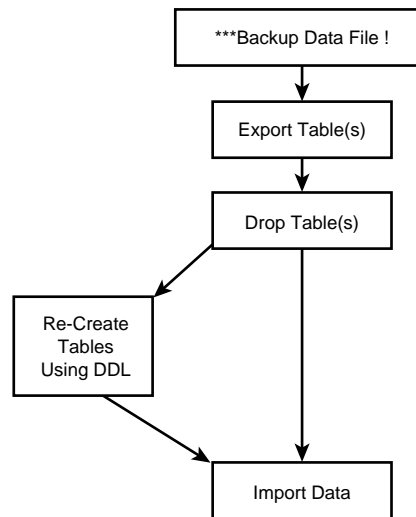
Before you explore the techniques for performing proper extent compression and de-fragmentation, let's look at a quick and dirty interim solution for objects that are nearing their maximum number of extents. When I create an object, I do not like to set the `maxextents` parameter at the maximum for the operating system. I like to set it a little lower. The difference between the operating system maximum and my settings is a safety margin that I keep for emergencies. Then, if I am nearing my maximum number of extents and I cannot immediately take the tablespace offline for de-fragmentation or compression, I can merely increase the number of extents for that object using the appropriate `alter` command. This gets me by until I have the time to fix the problem (with the de-fragmentation process described in this chapter). For example, suppose I have an operating system limit of 256 extents. I typically set the `maxextents` parameter for my objects at, say, 200. If the `golf_score` table reaches 195 extents and I know that I will not be allowed to take that table offline for a week, I merely increase `maxextents` to 225 with the following command:

```
alter table golf_score storage (maxextents 225);
```

The general concepts for de-fragmentation and extent compression are basically the same. You make a backup of the tablespace that you are going to work with (because you could make a mistake and you are going to do a deletion here). It's a good idea to do a complete backup of the database whenever it is feasible. Once your backup is complete, you follow one of two paths, depending on whether you are compressing indexes or tables.

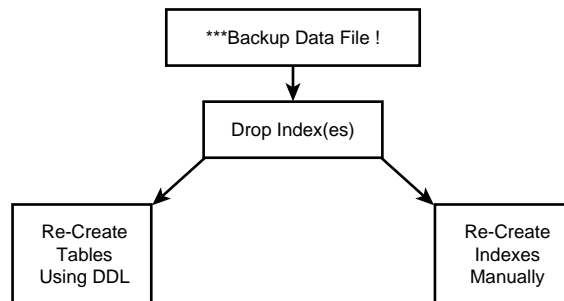
If you are working with tables (or a tablespace containing tables), you export the table or tables that you are going to compress. This can be a pain if there is a large number of tables in the tablespace that you are going to de-fragment, but Oracle does not provide the `export tablespace` option. Choose the `compress extents` option on the Export utility when performing this export and always select the options of exporting grants and so forth. Once you complete the export, you delete each table from the tablespace. You then run the Import utility to bring the tables back into the database where they will now occupy a single contiguous extent (except, of course, if they are larger than the maximum size of the extents for your system, in which case they will occupy the minimum number of extents possible). One caveat: if any of the tables that you are compressing serve as foreign keys for other tables that are not being compressed, you have to disable these foreign keys (`ALTER TABLE table-name DISABLE constraint-name`) during the dropping process and then re-enable them (`ALTER TABLE table-name ENABLE constraint-name` after you are done). Referential integrity can make life a little bit tricky for the DBA. The basic de-fragmentation process is shown in Figure 35.4.

Figure 35.4.
Process for compressing
extents of tables.



If you are compressing indexes, this is a much simpler process. The first thing that you have to do is perform your backup. Once that is done, you have to figure out what the structure of the indexes is. If you have saved your DDL (see the section titled “Keeping the Data Definition Language (DDL)” later in this chapter), all you have to do is modify these files to contain the new storage parameters, which will reduce the number of extents (if needed). Because the index is always based on the data contained in the table and you are not going to delete the table, you can simply drop the index or indexes and re-create them with the new sizing parameters. If you are trying to de-fragment the tablespace, you need to drop all the indexes first and then re-create them. Be sure that your rollback segments and temporary spaces are sufficient before creating any really large indexes. This basic process is shown in Figure 35.5.

Figure 35.5.
De-fragmenting and
compressing indexes.



Finally, let's say you want to de-fragment a tablespace that contains both indexes and tables. To do this you follow the procedure for the tables. This works because the indexes are exported with their associated tables. When you drop the table, its indexes are dropped and when you re-create the tables on import, their indexes are re-created.

Tip

De-fragmentation and compression are serious database surgery operations—treat them as such in terms of planning and double-checking.

A few final comments about these procedures: I always treat de-fragmentation and compression as serious surgery on my database. Any time I issue a `delete` command, I double-check to be sure that I have a backup of the data that I am deleting (just in case my export file becomes corrupted or I type the wrong table name when performing the export). I also double-check my command lines before I hit that Enter

key. Finally, I always try to do these after normal business hours. This may be the only time that I am allowed to take this data offline from the users. It is also helpful to do serious work during times when it is unlikely that you will be interrupted with telephone calls and other distractions—concentration is important here.

ALTERNATIVES: REDUCING DATA STORAGE

Recall the discussion earlier about deleting tables that are no longer required as a means of avoiding adding data storage space. This is the clean and simple method—the data in a table either all stays or it all goes. There are two alternatives that you should at least consider when you are faced with the challenge of reducing your data storage to make room for new data. The first alternative is to purge older data in your existing tables while keeping the newer stuff. The other is to steal space from other tablespaces to meet your needs.

First, let's look at purging data from your tables. Most production environments establish data retention requirements, which may be based on legal requirements or company policy. Although statisticians may prefer to have all of the data ever collected for their use, it comes down to a practical tradeoff as to how much you are willing to pay for disk space. Some places even prefer to get rid of old data to avoid having that data used against them if litigation occurs after the legal retention requirements for the data expire. The basic concept here is that you selectively remove rows that meet certain criteria (for example, the data entered is greater than five years old). You have the choice of placing the data in a permanent storage format (for example, a flat file that you can load back in using SQL*Loader) or just deleting it forever. When presented with an inability to add data files due to a lack of disk capacity (and you cannot buy new disks) and an inability to delete any of the existing data tables, you may go back to the users with the alternative of getting rid of some of their older data.

The other alternative to living within your existing disk capacity is stealing space from other tablespaces. This can be an alternative when you have a few user data tablespaces that are completely full and others that have plenty of space available. This will be a bit of a pain for you in that you will have to export all the data in the tablespaces that are relatively empty, drop and re-create the tablespace at the new, smaller size, and then import the data back into this tablespace. You then can create a data file with the space that you just freed up to add to the tablespace that was having the space problem. One note: Oracle typically does not delete the data files after you drop a tablespace. You have to go in at the operating system level and delete these files manually (do a complete backup and double-check your commands before hitting the Enter key).

KEEPING THE DATA DEFINITION LANGUAGE (DDL)

Another good idea when managing a database is to have a copy of the data definition language (DDL) for the objects that are stored within the database. These are the `create table`, `create index`, and `create view` commands that the developers used to create these objects. There are a number of reasons to have these scripts:

- ♦ It makes building the production instance a more controlled process because you will use the same scripts that were used in your test instance. You do not have the chance of running into typing errors or forgetting a field. You may alter the size of the extents before going to production, but at least the bulk of the script remains the same.
- ♦ You need this to move tables between instances without having to type the full DDL again manually at the command line. When you export a table, Import generally puts it back where you exported it. There are ways to force it into a new tablespace on import, but this can be a bit tricky. It is easiest to export the object, drop the object, and re-create it with a modified DDL script that uses the new tablespace.
- ♦ It is very useful for indexes, where you merely drop and re-create them. Because you never really see indexes, you usually do not remember on which fields they are located. This helps you remember.

The process for keeping DDL scripts is actually simple. You do not create tables interactively; rather, you always make and save the scripts before running them to create the objects. You then have to find a controlled location (or locations in the case where you want to separate development structures from production structures) to store the scripts until you need them. It actually is quite a simple process. You can store a script for any database object from tablespaces down to indexes and views.

SUMMARY

This chapter presented approaches to solving some of the more common problems with storing data within your Oracle instances. There can be many more obscure problems related to such things as bugs in the Oracle software, but you really need to contact Oracle support to find the correct solutions to these problems. They can even require specific solutions for specific operating systems. The key is to understand the basic storage problems presented in this chapter; they will make up the vast majority (well over 90 percent) of the problems that are typically encountered with Oracle storage.

The next chapter explores another set of problems with your Oracle instance—crashes of the instance or user processes. These are much less frequent in Oracle than space problems. Over 95 percent of the problems that I have run across are hardware-based (disk drive failures), space problems (these are by far the most common), and little things such as permission problems and users forgetting their passwords. I have run across some nasty crashes that took a lot of work with Oracle support, software patches, and even some work-arounds to solve. However, they are relatively infrequent and usually crop up only when you are dealing with a new version of Oracle.

- 
- Tracing the Problem
 - Log Files Can Help
 - Operating System Conflicts
 - Expanding Oracle Resources for Applications
 - Reducing Oracle Resources for Applications
 - When to Call Oracle

CHAPTER 36

Instance and Application Crashes

Hold everything! If you have instances or applications crashing, you may have some really serious problems. I am not just talking about applications that bomb out because they were coded with the table name misspelled. I mean those error messages that you've never even heard of before. This is especially true of instance crashes (where one or more of the Oracle background processes stop functioning). No matter how poorly the applications are written, these processes should stay up and running. If they come crashing down, it is likely that you have found a software bug. This usually requires support from Oracle to fix the problem.

Warning

If you have instances or applications that are crashing, you may have some really serious problems.

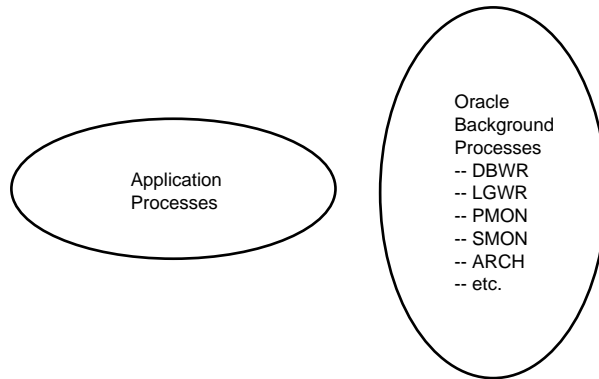
Now that I have your attention, what does this chapter cover? First, you learn how to find out what was happening when the crash occurred. Next, you explore some of the interactions between Oracle and the operating system that can cause these problems. Then you look at ways that you can either increase Oracle resources to help applications run or decrease the amount of resources that the applications require. Finally, I give a few words of advice as to when to call Oracle about the problem.

TRACING THE PROBLEM

There are really two problems in this chapter, and it is important to separate them from the start (see Figure 36.1). The first problem deals with crashes of the Oracle background processes (database writer, log writer, the SQL*Net listeners, and so forth). These simply should not occur. Although it is possible that the operating system administrator or DBA can do things that cause the processes to crash (such as loading a new version of the operating system or RDBMS that has bugs in it), it could be an indication that there is a problem in the Oracle software itself. You are not in a position to fix the RDBMS software and therefore you may need Oracle support.

The other problem set involves the crashing of user application processes. This could indicate a problem with the Oracle software, but it is likely an indication that the application has been designed in a manner that is incompatible with your Oracle instance. It could just be a bad design (that is, you add a million records before you commit) or it could be an indication that some database tuning is required to enable your users to do what they need to get done. Your challenge is to figure out which of these two possibilities is true.

Figure 36.1.
Different process crash
problems.



The first part of the problem solving process is to trace down exactly what is causing the problem. This usually starts with a review of the screen error messages and any log files that are produced. (The range of log files that are available is discussed in the next section.)

I usually like to print the applicable sections of the log files (when this is reasonable—some dumps can be quite extensive) and screen prints of any error messages and spread them out on my desk. If you do this, be sure to look for database trace files and error messages in the alert log even if you received an application error message. Once you have all the data that is available, you should look at it and determine whether you have sufficient information to know what the problem really is. Here is a sample screen error message that may indicate a crashed instance:

```
SQL*Plus: Release 3.1.3.5.4 - Production on Mon Aug 07 21:01:47 1995
```

```
Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.
```

```
ERROR: ORA-01034: ORACLE not available
```

```
ORA-09243: smsget: error attaching to SGA
```

```
OSD-01104: unable to attach to SGA: SGA does not exist
```

```
Enter user-name:
```

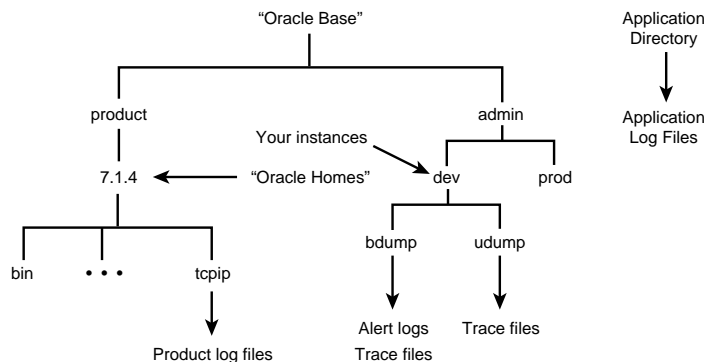
If you do not understand the problem, you should look at your list of DBA views to see what additional information you can gain by querying the database. Perhaps you should run some of your tuning, configuration, and utilization reports to see whether there is anything unusual or out of specification. Do all that you can, within reason, to be able to isolate the problem. Even if you have to call Oracle technical support, it is helpful to have as much information as possible. You can cut hours off of the time it takes to get the problem solved if you collect all your data and can precisely describe the problem to the support staff members.

Here is where that intimate knowledge of where things are located in your database and your applications comes in handy. You should know where all your log, data, control, and initialization files are located. Make a quick drawing showing their locations and keep this drawing handy in a binder on your desk or post it on your wall. A little time investment up front when things are calm can save a lot of headache when things get hot. You should also have an understanding of where all of the application stuff is located, including any application log files. When a database error occurs, you are usually the one who gets called in and you do not want to sit around waiting for a developer so that you can get access to the log files.

LOG FILES CAN HELP

Let's go through another quick review of the log files. The common log file resources are shown in Figure 36.2.

Figure 36.2.
Common log file
resources for problem
solving.



If your database stays up (the Oracle background processes do not crash), the first place that you should look is the log files or screen prints from the applications themselves. If your developers are making a lot of applications that run in batch mode but do not produce log files, you really should have a talk with them. It is usually not a difficult task for them to modify their batch applications to produce log files and it can help immensely when a problem occurs. People, like Oracle technical support staff, cannot be very helpful when you cannot accurately describe the problem.

If your database crashes, you should expect to see a series of trace files that were written at the time of the crash (look at the date/time stamp on the files to figure out which ones correspond to this problem). You will usually find more than one, because when one of the background processes crashes the other processes detect this and bring themselves down. Your challenge is to look for the one with the original error message and not the ones that say that they detected another process going down. The trace files are located under the bdump directory specific to your instance in the Oracle admin hierarchy.

You also can find useful information in the alert log for your instance. This, too, is located in the bdump directory for that instance. There will not always be a record of the crash in this file, but you may get some feel as to what was happening before the crash. One classic example is to check for the log file switches that are recorded in this file. If you see a high number of log file switches, this indicates that the database was performing a large amount of updates just before the crash. This is valuable information that you should factor into your analysis and also provide to Oracle support if you call them. Here is an example of a log file that shows an instance failing to come up (I renamed one of the data files so that it was unable to find one of its key files for startup):

Starting up ORACLE RDBMS Version: 7.1.4.1.0.

System parameters with non-default values:

```
license_max_sessions      = 25
shared_pool_size          = 3499008
control_files              = %RDBMS71_CONTROL%\ctl1.ora, %RDBMS71_ARCHIVE%\ctl1.ora
log_archive_dest           = %RDBMS71_ARCHIVE%
remote_login_passwordfile = SHARED
mts_servers                = 0
mts_max_servers            = 0
mts_max_dispatchers        = 0
audit_trail                = DB
sort_area_retained_size    = 65536
db_name                    = oracle
background_dump_dest       = %RDBMS71%\trace
user_dump_dest             = %RDBMS71%\trace
```

Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Mon Aug 07 21:12:59 1995

PMON started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Mon Aug 07 21:12:59 1995

DBWR started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Mon Aug 07 21:13:00 1995

LGWR started
Dump file C:\ORAWIN\RDBMS71\trace\ALERT.LOG
Mon Aug 07 21:13:01 1995

ALTER DATABASE oracle MOUNT
Mon Aug 07 21:13:01 1995

Completed: ALTER DATABASE oracle MOUNT
Mon Aug 07 21:13:01 1995

ALTER DATABASE NOARCHIVELOG
Mon Aug 07 21:13:02 1995

Errors in file C:\ORAWIN\RDBMS71\trace\DBWR.TRC:
ORA-01157: cannot identify data file 5 - file not found

```
ORA-01110: data file 5: 'c:\orawin\dbs\system2.dbf'  
ORA-09202: sfifi: error identifying file  
OSD-01002: unable to open file (OS 2)
```

```
Completed: ALTER DATABASE NOARCHIVELOG  
Mon Aug 07 21:13:02 1995
```

```
ALTER DATABASE oracle OPEN  
Mon Aug 07 21:13:02 1995
```

```
Errors in file C:\ORAWIN\RDBMS71\trace\DBWR.TRC:  
ORA-01157: cannot identify data file 5 - file not found  
ORA-01110: data file 5: 'c:\orawin\dbs\system2.dbf'  
ORA-09202: sfifi: error identifying file  
OSD-01002: unable to open file (OS 2)
```

```
ORA-1157 signalled during: ALTER DATABASE oracle OPEN ...
```

```
Shutting down instance (abort)
```

```
License high water mark = 1
```

Here is an example of a trace file—in this case, a database writer (DBWR) process trace file related to this problem:

```
MS-WINDOWS Version 3.10  
Mon Aug 07 21:13:02 1995
```

```
*** SESSION ID:(2.1)  
ORA-01157: cannot identify data file 5 - file not found  
ORA-01110: data file 5: 'c:\orawin\dbs\system2.dbf'  
ORA-09202: sfifi: error identifying file  
OSD-01002: unable to open file (OS 2)  
ORA-01157: cannot identify data file 5 - file not found  
ORA-01110: data file 5: 'c:\orawin\dbs\system2.dbf'  
ORA-09202: sfifi: error identifying file  
OSD-01002: unable to open file (OS 2)  
Dump file C:\ORAWIN\RDBMS71\trace\DBWR.TRC
```

The alert log and background process trace files provide information when the main RDBMS processes crash. Certain applications that are not specific to a given Oracle instance (such as SQL*Net, which is available to all instances on a given computer system) have separate log files. They are usually located in the directory hierarchy of the product itself (tcpip for the SQL*Net TCP/IP product). These are the ones that are usually overlooked when searching for information.

One final note about log and trace files: Under routine maintenance, you should routinely clean out these files. This can be very helpful when you encounter a problem. It can be a real pain to try to read through a 1M log that has not been cleaned out since you started the instance. (I had to do that once with a SQL*Net log that no one knew about.) Regular cleanup can make your work much simpler. However, if you have a problem that is still pending, you may want to keep the trace and log

files around until the problem is resolved. I like to make a subdirectory called something like “retain” somewhere in the admin hierarchy. When a real problem comes up, I copy (not move) the applicable files to this directory so that I have a copy even after the log purge routines kick in.

OPERATING SYSTEM CONFLICTS

The most confusing types of problems that can occur with an Oracle database are those that involve a conflict between your version of Oracle and your version of the operating system or its utilities. This is because these types of conflicts can confuse most of the Oracle error logging procedures. From your point of view as a DBA, the Oracle instance “just died.” In most of the places that I have been, the burden of proof is placed on the Oracle DBA.

It is impossible to predict all the scenarios that could result from these types of problems. Here are a few of the ones that I have run across so you can get a general idea:

- ◆ A disk drive array intermittently lost connection with the computer bus. Because the entire Oracle system was on this drive array (including the log files), it would just appear as though Oracle went away—no log files, no screen error, nothing. I had to collect a lot of data and point out that the system error logging utilities were turned off to prove that it was not an Oracle problem. Once the system error logs were turned on, they recorded a flood of errors to this disk array. The array was eventually replaced. This was, by far, the ugliest problem that I ever had to deal with as a DBA.
- ◆ We had just upgraded the operating system and Oracle software. The Oracle instance would crash, starting with the database writer process. I recorded the error messages and called them into Oracle. They had a record of this problem before with this operating system and sent me a relatively small patch through Internet mail that I received and applied within two hours.
- ◆ We started to receive intermittent ORA-0600 messages and crashes of user jobs when they were performing updates to the database. I saw a lot of activity in the redo log files just before the crash. I called in the ORA-0600 message (it has a set of internal error codes after it to help identify the specific problem that Oracle technical support needs) and discussed the log activity. Technical support looked it up and indicated that it was a known bug that required a major rework of the logging system that would not be completed until version 7.2. I increased the size of the redo logs to reduce the number of log switches and used this work-around to stop the error messages until 7.2 becomes available.

Most of these types of problems can be confusing when you look at the data presented. It is especially important here to record every scrap of detail in an accessible form before you call Oracle technical support. If you can give them the specifics, it can be a relatively quick (although usually not painless) process of tracking down the error. Odds are that someone had the problem before. Keep copies of everything (including core dumps and those very long trace files that are sometimes produced) so that you can answer the hard questions that may arise.

Tip

Always record every scrap of information available to you and have it ready when you call Oracle support. It can save a lot of time when the Oracle staffer tries to diagnose your problem and get you backup from thousands of miles away. It also presents a good image of your abilities as a DBA.

EXPANDING ORACLE RESOURCES FOR APPLICATIONS

A high percentage of application crashes are the result of insufficient resources within the Oracle system. Classic examples include insufficient rollback segments and cursors within your system. Many solutions will be found in the messages and codes manuals. When you look up the error message number, it tells you to increase this or adjust that. In other cases, you call Oracle and they indicate what to do.

Adjustments to Oracle resources come in two basic flavors—those that you can adjust on an interactive basis with the database and those that require a shutdown and startup of the instance. The interactive ones are usually easiest. An example of this is adjusting the size of the optimal storage parameter for the rollback segments. The ones that require a shutdown are usually those found in the initialization files. Here, you edit the parameter and then restart Oracle to have the parameter take effect.

The real trick when adjusting one of these tuning parameters is knowing to what setting to adjust the values. I always keep a configuration report on my desk that lists the settings that are in effect for the tunable parameters in the Oracle instance (including those that you are using the default values for because you do not have them in your initialization file). If you call Oracle support and tell them what your current setting is, it can usually recommend a value to which to adjust your tuning parameter. If you do not call Oracle support, always make the minimum adjustment possible to solve your problems. This may take several iterations, but it prevents you from wasting a lot of memory, which is generally tight on Oracle systems.

REDUCING ORACLE RESOURCES FOR APPLICATIONS

Another way of looking at resource problems is to adjust the applications so that they require less in the way of resources. One of the most common examples is the frequency with which the applications commit transactions. I have seen developers put in thousands of rows and then wonder why they ran out of rollback space. Developers are busy people too, of course, and you should try to provide as much support as possible from within the Oracle world. However, you have limitations in resources (for example, memory boards), and they cost your company a lot of money. At least consider whether the demand placed on the system is reasonable before you just throw more memory at it.

The keys to determining whether the problem lies with insufficient resources or an inefficient design are knowledge of Oracle's internals (remember that difficult technical stuff at the beginning of this book) and a knowledge of your applications. Always ask a few questions about what the application is doing before you try to solve a problem. Avoid taking a tone that the application is defective. These questions are useful even when the problem lies with your database.

This is yet another one of those situations in which there are an enormous number of possibilities that might arise. Here are some problems that were solved by tweaking the applications rather than the database:

- ◆ A number of applications commit only the end of a very long loop that inserts or updates hundreds or thousands of records. If you get an error indicating insufficient rollback space, you should work with your developers to see if it is possible to commit a little more frequently.
- ◆ A number of queries exceed the capacity of the temporary tablespace. When I looked at the output, there were not many rows being returned. When we looked at how the query was being executed, it was using the merge-join execution plan and pulling rows from a very large data table before sorting them against the relatively small number of rows in a lookup table. Anyway, with a little tweak to the syntax of the `select` statement, we were able to avoid increasing the temporary tablespace size and made the query much faster.

WHEN TO CALL ORACLE

Once again we come to another decision call that you have to make as an Oracle DBA. When do you admit that you cannot figure out what the solution to your problems are and call Oracle technical support? You could wait until you can go no farther on your own. This may slow down the problem resolution process, though,

because Oracle usually has seen the problem before and has the approved solution sitting in its database waiting to be called.

Let's go over the pros and cons of calling technical support when you have a problem. These are the factors that you weigh when you make your decision:

- ◆ They are the only ones who have the internal technical bulletins and software patches that are needed to fix some problems.
- ◆ They have a database that contains all the bugs and fixes that have been called into them. The odds are that if you can describe the problem precisely, it is in that database waiting for you to call in and ask.
- ◆ Your time is valuable. You may have a dozen other issues to work on and any help that you can get from the outside can be important.
- ◆ It may take you quite a long time to experiment and determine the answer to the problem that someone has already solved. It can be quicker in these cases to get the answer from the vendor rather than figure it out for yourself.
- ◆ You paid for it. You don't want to bother them with a lot of little questions that you should already know the answers to, but you should not have to kill yourself trying to find answers to harder questions that are covered under your support contract.

There are some reasons why you may not want to call Oracle technical support:

- ◆ You may not have a support contract in place, or your contract may not provide support in the middle of the night when you are trying to solve the problem. In this case, you are on your own—at least until regular business hours.
- ◆ Sometimes it may take hours or even days for Oracle to get back to you. Depending on the priority of your system (is it a down production system, a problem in a system that is operational, or a routine question?), you may have to wait. Sometimes you may want to call in the problem once you realize that you will probably need help, but continue to work on it yourself in case it takes a while for you to have your call returned.
- ◆ You know your system far better than any outsiders. It is a challenge for someone in California or Florida to gather enough information from you over the phone to nail down what your precise problem is.

You have your own threshold determining when you will make the call for outside support. These are the times when I call: First, I call whenever I look up the error message and it tells me to call (as in the case of ORA-0600 messages). That is relatively easy. I also call when I have unusual crashes that provide little information and the error messages that I do have do not tell me what to do. For installations, I call whenever I have tried all the basic tricks in the Installation and Configuration Guide (tempered by the information in the README file) and I still cannot get the installation done. Probably half the calls that I have made to technical support have been during or just after installing new software on the system. Finally, when I try what is suggested in the errors and codes manual but it does not help the problem, I call. Everyone has a different threshold for deciding when to call. New DBAs may call more frequently than oldtimers who have been down the road before.

SUMMARY

This chapter addressed one of the more serious problems that can crop up in your Oracle world—crashes. These crashes need to be classified as to whether the entire Oracle instance shut down (which is usually the more serious situation) or a single user process bombed out. Log files and other resources that you can use to help track down problems were explored—remember, your developers should be making log files for their batch applications to help you. Next, this chapter discussed perhaps the most difficult problems: those associated with the interaction of Oracle with the operating system. There is a tradeoff between adding Oracle resources and having developers make their applications more efficient. Finally, you learned when to consider calling Oracle for some technical support.

-
- A detailed black and white illustration of a celestial globe. The globe is tilted, showing the zodiac signs Gemini, Cancer, and Leo. Gemini is depicted as two figures, Cancer as a scorpion, and Leo as a lion. A large, bold 'MAR' (March) is written across the lower left. A band labeled 'TEMPERATE' runs diagonally across the globe. The background features a landscape with a city, a river, and a mountain. The text 'I. Terra', 'S. Michael', 'S. Peter', 'S. Paul', 'S. John', 'S. James', 'S. Andrew', 'S. Thomas', 'S. Philip', 'S. James', 'S. John', 'S. Paul', 'S. Peter', 'S. Michael', 'S. Andrew', 'S. Thomas', 'S. Philip', 'S. James', 'S. John', 'S. Paul', 'S. Peter', 'S. Michael' is visible on the right side.

[illegible]

When the Database Is Too Slow

Remember the story about the boy who cried wolf? There was a little boy in a village who tended sheep just outside of town. He found it very lonely to sit out there all day. One day, when he was especially lonely, he decided to cry out “Wolf!” which brought all the members of the village out to help protect the flock. When they arrived, he said that the wolf fled just before they got there. The next day, he decided to cry out “Wolf!” again and once again the villagers all came out and he was less lonely. Eventually, as he continued to do this, fewer and fewer villagers came out. One day a real wolf came and the little boy cried out “Wolf!” loudly over and over again, but no one came. They all thought the little boy was just lonely again. The wolf ate all the sheep and the little boy, too.

With many Oracle instances, the users will never be satisfied with response time. They will cry out that the database is “too slow.” You may have just put in a new system enabling them to issue queries that take ten seconds and replace an overnight batch system. However, the users will soon start complaining that it should not take ten seconds for this query, so you may become accustomed to a continuous chorus of “too slow” and be tempted not to pay attention to these complaints. I guess the moral of the wolf story for the DBA is that just because the users always complain about performance does not mean that they are always wrong.

Chapter 32, on tuning, presented some of the basics of determining whether the Oracle database is internally tuned to provide a good throughput. This chapter presents some of the other factors that may lead to degraded response time. The goal is to provide you with a method for assessing the performance of your system in a scientific manner and concluding whether your system is performing up to its capabilities. A side benefit is that you will have facts and figures that you can use as ammunition when people start to complain. It will also give you data as to what you might want to change when there are real problems.

WHEN IS A DATABASE TOO SLOW?

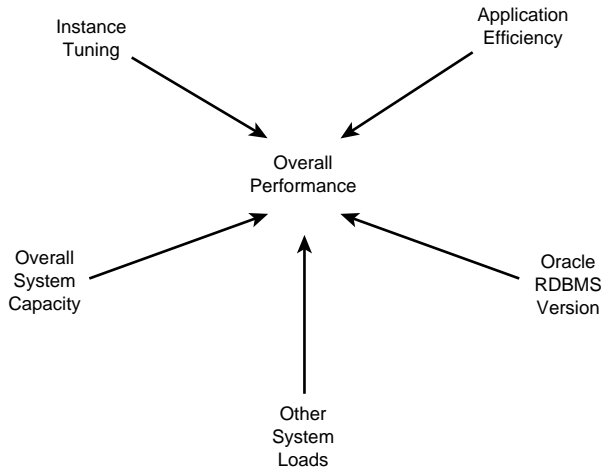
A pure university theorist would probably say that the database is too slow whenever any component within the system is not operating at its maximum capacity. I am not a university theorist. I look for some level of efficiency at which the cost is worth the gain. You need some standard against which you can measure your performance.

First, you need to understand the factors that affect the efficiency of your system. Again, there are many ways that you can divide this and many small factors that you can consider. The following are factors affecting performance (see Figure 37.1):

- ♦ Tuning of your instance (for example, your memory areas)
- ♦ Efficiency of your applications

- ◆ Overall capacity of the system
- ◆ Version of your Oracle RDBMS software
- ◆ Other loads on the system

Figure 37.1.
Factors affecting the
performance of Oracle
applications.



Let's start with something that you probably cannot control—the other loads on the system. If you have a computer system to yourself as a server for Oracle, you do not need to worry about this. However, if you share a server, you need to determine whether the problem is with your applications or the other applications. It can save you a lot of time you would waste chasing after something that is not there. The easiest test is to schedule a time when you can take your system down for a short period of time. Then measure the loading on the system and compare this figure to the loading that you normally have at that time of day. For example, if you have a UNIX system that is running at 0% idle time normally and when you take your instance down the system runs at 2% idle, it is likely that you are not the big problem on that system.

Another factor that is somewhat beyond your control is the overall capacity of the system. It is a combination of the CPUs, disk drives, and memory boards that are installed. You can usually request more in terms of these items, but don't expect them to be delivered overnight. Because these are capital expenses, expect to have to exhaust all other alternatives before you get to buy additional capacity.

Now on to a factor that you can control—tuning of your instance. In Chapter 32, you learned some suggested analyses to run in the form of a tuning report. This report catches the majority of tuning problems that can arise in an instance and also provides suggested ranges of values for the parameters that Oracle recommends. This report and the Oracle references are good enough to prove to people whether

the tuning of the instance is adequate. For example, you may find that the only parameter that exceeds the recommended values is `database buffer cache` being about twice the normal value. You may choose to increase `database buffer cache` from 5 to 7M and see what happens to performance and the tuning report values.

The next area, application efficiency, unfortunately lacks any simple numerical standards. Each application varies in complexity and many other factors; therefore, it is difficult to compare one application to another for efficiency. You have to rely on some experience to understand which queries seem to be running a little bit too long. Once you identify potential problems, you need to try some alternative algorithms and execution plans to see whether you can improve performance. I haven't heard of any absolute rules and simple formulas (perhaps that is why everyone doesn't become a DBA—there is obviously a limit to the number of crazy people in the world).

The final area that you can work on is upgrading the version of your RDBMS. Each release tends to provide a little bit more in the way of functionality and more options to improve performance. You have to pick when to upgrade based on whether these features are of use to you. For example, Oracle 7.1 introduced parallel query processes where you can have a number of system processes working on your query to speed performance. This enhancement is designed to work on computers that have more than one CPU. If you do not have more than one CPU, you probably will not benefit from this particular upgrade. The same logic can be applied to operating system upgrades. This is something into which you usually will have some input.

MANAGING USER EXPECTATIONS

Perhaps all these problems start when information systems folks or users try to sell new system upgrade projects. In the typical scenario, the project sponsor promises the sun, moon, and stars in order to get approval for the project. It probably does improve performance compared to most legacy applications. However, end users are used to the speed and performance of their word processors and spreadsheets. These tools process little bits of information. When users query a database, they may be searching through thousands or millions of records of data in multiple tables to come up with a few lines of answers. However, all the users see is that two lines of output took twenty seconds.

Your challenge is to ensure that your user community has reasonable expectations for the database. This is definitely a challenge. Users are not into the technical details of database query algorithms or network communications protocols. You have to ensure that their expectations for performance are reasonable.

How do you take on such a monumental task? Do your homework! Run performance tuning reports and see what is going on with those parts of the application that are

performing poorly. Gather some data and at least try to improve your system's performance. Many users do not trust computer types—perhaps based on bad experiences with some of the more arrogant of our brethren back in the days of the all-powerful data center. You have to convince them with some test results that you have at least given it a try.

Another technique that is sometimes useful is to get an outsider to look at your database. I have occasionally worked on contracts where the local DBA was doing things correctly, but no one believed him. When an outsider strolls in and says things are going well, they suddenly believe it. It may seem strange, but there is sometimes a mystique about some “expert” from the outside that appeals to certain people.

You must remember to communicate. There are a lot of DBAs who work very hard. They do what they can and work long hours, but no one ever hears about it. It can be very useful if you occasionally get up and talk to your developers and a few of your key users. A lot of computer types are better with a terminal than with people. Think of this as yet another challenge that faces you. It can help your relationship with these folks and also deepen your understanding of the applications and users, which can come in handy when problems come up or it is planning time.

CHECKING TUNING OF THE DATABASE

Database tuning is the one factor that is usually most closely associated with the DBA when it comes to performance. Sometimes you are assigned to help the developers with their applications, but typically it is their job to ensure that the applications are efficient. Therefore, you should ensure that your instances are properly tuned. This is not as hard a task as it might sound. There is a standard query set included on the disk that comes with this book. It represents not just my opinions, but those of the Oracle developers as to what is a properly tuned instance. There are a few folks around who are technically knowledgeable enough about Oracle to be able to argue these types of details with the developers, but odds are you won't have any of them in your user community.

It might be a good idea to review the basics of tuning. Start by measuring performance: issue the queries contained in the tuning report. Then compare the results for your instance with the standard criteria (goals) for a well-tuned instance. When you find a performance indicator that is out of specification, adjust the Oracle tuning parameter specified on the report. This typically is an iterative process wherein you bump up the parameter in small chunks until you get the performance indicator to where you want it.

Most of the internal counters used to measure Oracle performance parameters are reset when you start up your instance. Therefore, you need to give your instance some time to run under normal operating conditions to ensure that the data you

collect is good. This is true for the tuning report on the enclosed disk and any other queries or tools that you acquire for performance monitoring.

What about all of those other performance parameters? Well, they did not make the hot list from Oracle that was used to make up the enclosed scripts. They are out there and could affect database performance, but I have not found a good set of criteria against which to compare the actual values; therefore, it would be difficult to say what the results indicated. The only exception to this is a series of undocumented parameters Oracle occasionally recommends that you set if you have an unusual instance. When I was working with that 135G data warehouse, we set about a half-dozen parameters that could not be found in any of the published literature. Perhaps Oracle thinks that some of these parameters are too powerful or perhaps they can mess up the entire database if set incorrectly. It was a case where you just had to trust them, and it did seem to improve performance when those parameters were used.

There are people who will come in on a consulting basis and perform some very detailed tuning analyses. Most of the large national firms, including Oracle, and a number of local firms love to talk about this kind of business. Your goal in this case is to get an assessment of the performance of your database and applications. You also want to get specific recommendations as to what you should do to improve performance.

Finally, there are a number of third-party tools on the market, scripts that you can obtain from bulletin boards, and so forth. There are even some that are provided with your Oracle software. The `bstat` and `estat` utilities enable you to gather statistics across a period of time (`bstat` gathers beginning stats and `estat` gathers ending statistics). This can be useful for measuring problems over peak periods. With all these tools, your goal is to ensure that you have some criteria against which you can assess your performance. In the end, you should tune your instance to the point where your performance monitoring reports show that you have a well-tuned instance.

APPLICATION TUNING

Now on to the more difficult subject—application tuning. To properly tune applications, you need to be familiar with the software itself. For example, an application can look up lists of legal values every time it makes a call to one of your data tables, or you can look up the legal values at the beginning and store them in memory variables. You also can adjust your SQL so that you use a different execution plan. I have seen slight changes that modify the execution plan to turn 30-minute queries into queries that execute in less than 30 seconds. (Talk about the opportunity to be a hero!)

The first thing to look at is the basic algorithms used. Do they make lookup data as a series of multiple queries to the database or do they call in the information needed in one fell swoop? Look also to see whether they commit frequently or try to wait until the end of the entire program to perform these commits. Both of these circumstances can cause Oracle to demand an excessive amount of resources and thereby degrade the overall performance of the database.

The second thing to look at is the SQL used in the `select` statements. Run it manually with the `trace` utility turned on. There is more detail about this in Chapter 40, but for now, remember that you need to experiment with different execution plans to get the one that maximizes your performance. With some experience, you will come to recognize execution plans that are really poor performers (merge-join, for example). You will also understand how to alter the execution plan with the use of hints or the cost-based optimizer.

Finally, look at how the data is accessed. If you set up a number of views for the database, it may become convenient for your developers always to use those views without thinking whether they could get the data directly from a table (which normally is faster). You also need to look at the order in which the fields are listed to ensure that you will take advantage of any indexes that are available on that table. Whenever possible, use indexes. Not only are they much faster for even small tables, their performance degrades much less rapidly when you move to those enormous tables that you've been dreaming about creating.

You should now have a general understanding about the types of things that you look at in a program to try and improve its performance. Experienced application tuners can look at the code and instinctively know what to do to get that improved performance. The rest of us rely on guided trial and error to find the right combination of factors.

WHEN ADDITIONAL CAPACITY IS REQUIRED

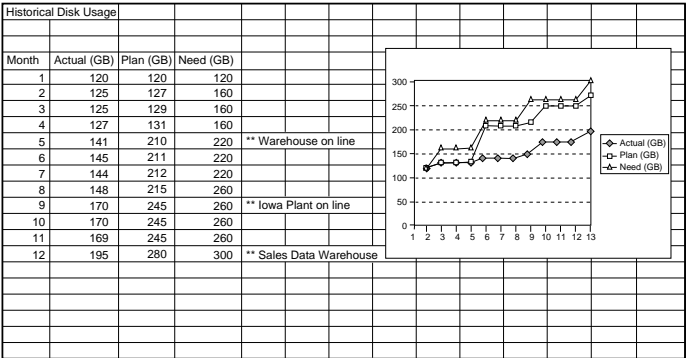
Recall that you may face a lot of opposition when the conclusion that you draw from your investigation of performance is that you need to buy additional capacity. Capital budgets are usually tight. However, you will probably get even more questioning and scrutiny when you detect the needs as part of troubleshooting as opposed to annual budget planning. People get upset when things do not go according to their budget plans. They are often placed in the position of deferring a planned upgrade to meet this emergency need. You have to acknowledge these feelings and ensure that you are very well prepared to defend your conclusions.

Chapter 33 provides some general guidelines for presenting the bad news to management and users. Let's review some of these here before you explore a few more thoughts on the subject. When you have a problem, you know the resource that

you need (memory, disk, or processing capacity). However, take a look at the overall situation before you make any requests. A problem with memory areas not being large enough may be an indication that your tables are getting larger than expected. You may be heading for a disk increase in a matter of weeks or months. It is usually easier and makes you look more thorough if you can present the overall picture of the problem in one sitting for management. Besides, you probably don't want to have to go through these briefings very often.

With that said, it is time to review the factors that you need to consider when assessing capacity problems (see Figure 37.2). This is useful because it will jog your thinking as to what it was that caused you to run out of resources. Was business just much better than expected so you ran out of space on the orders database? Was the new warehouse application actually much larger than was initially predicted? Because this will be one of the questions that you are likely to be asked, it is a good idea to be prepared to answer it.

Figure 37.2.
Factors affecting
capacity projections.



Being a creature of habit and pattern, I follow the same procedure when performing a capacity problem analysis that I use when preparing an annual capacity plan. I start with my collection of historical information showing how resources have been used to date. When analyzing problems, I add a line showing the original predicted growth rate for the resources. Finally, I add a third line showing capacity over time. For a sample of this type of graph, refer to Figure 33.1.

The first thing that you are looking for on this graph is where the actual usage started to deviate from planned usage. Your job is use this to figure out what caused the deviation. Often it is a combination of factors. Because it is usually impossible to give exact figures on how many bytes were from the warehouse application versus unexpectedly brisk business, you can usually get away with some rough estimates of the impacts (70 percent business, 30 percent warehouse). Your job is now half over. You have explained why things have changed since the last estimate.

Now comes the slightly more tricky exercise. You need to revise your capacity forecasts. You may say that you know you are 2G ahead of plan; therefore, please send a new 2G drive. Such a conclusion is usually wrong. You need to go back to your original calculations and modify them to accommodate your new trends. If business is 30 percent brisker than planned and you expect this to continue, you should multiply your growth figures for data related to orders by 30 percent. The same applies to future projections of the size of that warehouse application that was larger than expected. Perhaps it was larger because they added some static lookup tables. However, if the scope of the project crept up and you are storing more data about every item in the warehouse, you need to factor this into your future size projections for this application.

Your goal should be fixed in your mind. Take what you have learned about changes in capacity based on your actual data. Go back to the developers and users to ensure that your assumptions are still sound. Make the best estimate that you can based on these new numbers and then present them. This is where a spreadsheet comes in handy. It enables you to modify your numbers and recalculate the totals. You can even play what-if games to try out scenarios until you find the one that best fixes your current capacity problems.

One final comment about safety margins: You have just dropped a bit of bad news on management and users. They may have understood why these things have happened and appreciate the thoroughness of your work, but they probably did not like it. Perhaps now is the time to reconsider the concept of safety margins. Neither you nor your management enjoyed this little surprise that you just presented. Perhaps now is the time to consider whether, for example, a 20- or 30-percent safety margin for sizing tables is in order. You have just proven that the world does not always go according to plan. You can plan in a proactive manner or react to crises. A little extra disk or memory capacity will be used up in the not-too-distant future anyway, so it will not go to waste. Bring this up if you think people might be receptive.

SUMMARY

This chapter addressed the issue of databases that are perceived to be too slow. It's best to avoid arguing about what is too slow and instead break it down into a series of analyses that you can perform to determine overall efficiency. Always try to gather accepted performance goals for things such as the tuning report so that you can present a factual account as to whether the instance is operating within normal parameters. When it comes to tuning applications, things can get a little stickier because there is not a simple formula that you can use to determine efficiency. However, this chapter presented the things that you can look at and try to see whether you can improve performance (more on application tuning in Chapter 40).

Finally, you learned how to deal with the situation wherein your problem is a result of insufficient capacity. Hopefully, it will minimize the pain when you present your conclusions.

The next chapter is a summary—in the form of a checklist—of the tips on dealing with problems that are presented in this part of the book. It is compiled on a somewhat general level because Oracle problems and solutions are often specific to the given version of the Oracle products that you are using and perhaps even your operating system. The messages and codes manuals, along with the specific manuals for tools such as SQL*Net and the operation-specific manuals, provide you with the details on what to do with a given message. This checklist stresses overall methodology, which is sometimes lacking when you get into the technical references.



- Checklist

CHAPTER 38



A Troubleshooting Checklist

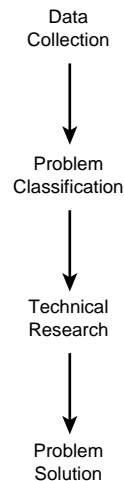
This chapter pulls together some of the pieces in the last couple of chapters and turns them into a checklist that can be used when you run into problems. Some of the more experienced DBAs may consider most of this to be second nature and therefore not need it. However, it will be especially useful to the new DBA for whom the troubleshooting process is a new experience. Before you jump into the checklist itself though, let's look at a graphical road map that shows you generally where you are going (see Figure 38.1). This checklist is on the enclosed disk so that you can copy and modify it to suit your own tastes and needs.

This checklist summarizes general advice and avoids specifics because it would be impossible to predict all situations and keep the checklist accurate for all versions of Oracle and all the operating systems under which Oracle works. Think of it as an overall picture of what is going on when troubleshooting a problem that rarely appears in the technical reference manuals (which tend to jump immediately into a list of error messages, arranged in numerical order or something like that).

As you can see from the figure, troubleshooting starts when a problem is detected. Collect as much data about the problem as possible and necessary before jumping into the solution. Many problems in Oracle can be different than they appear and you will feel more comfortable when you have the whole picture.

Next comes time to classify the problem. There are two forms of classification that you have to make. The first is determining what part of Oracle is acting up. The other part is determining where this problem falls in the grand priority scheme. After that, you go into the technical research phase where you look up error messages and possibly contact Oracle technical support to determine the best course of action for solving the problem. Finally, you get to take action to solve the problem.

Figure 38.1.
Overall picture of the
troubleshooting process.



CHECKLIST

PHASE 1: DATA COLLECTION

- ☐ Get a screen print of any error messages on the user screens.
- ☐ Get a copy of the application log file if produced by a batch application.
- ☐ Review the instance's alert log for any indications of activity or problems (instance and application crashes).
- ☐ Look for trace files that were produced around the time of the problem in the instance's bdump and udump directories (instance and application).
- ☐ Look for core dumps that were produced around the time of the problem in the instance's cdump directory (instance and application crashes).
- ☐ Make copies of any of the log and trace files that contain potentially useful information to a separate directory that is under your control.
- ☐ Run a database utilization report (space problems).
- ☐ Run a database tuning report (resource problems causing application or instance crashes).
- ☐ Run a configuration report (resource problems) so that you have a current picture of the settings for your tunable parameters.

PHASE 2: PROBLEM CLASSIFICATION

- ☐ Based on the data collected, make a first attempt at classifying the nature of the problem into one of the following categories:
 1. Space problems
 2. Oracle crashes (resource problems, software bugs, and so forth)
 3. Performance problems (the database is too slow)
 4. Hardware or operating system problemsYour best estimate: _____
- ☐ Classify the severity of the problem into one of three categories: routine task, problem, or *real* problem. Base your classification on a review of the following factors:
 1. Priority of project or system to the company
 2. Severity of the error (crash versus warning)

CHECKLIST

PHASE 2: PROBLEM CLASSIFICATION

3. Affect on general health of the system

4. Deadlines for project or user tasks

5. Company politics

6. Difficulty of the fix

Your best estimate: _____

PHASE 3: TECHNICAL RESEARCH

- ☐ Look up and write down the meanings and recommended actions for the various Oracle error messages that have appeared on the screen and in the log and trace files. These can be found in the messages and codes manual, the operating system-specific manual, and the specific product manuals (for example, SQL*Net).
- ☐ Look at the Server Administrator Guide and the server concepts manuals to gain more information about database object or constructs that you are not familiar with (for example, rollback segments).
- ☐ If you are baffled as to what to do, call any fellow DBAs who may have run into a similar situation.
- ☐ If you are baffled as to what to do, call Oracle technical support if you have a contract in place for their services. Note that you get post-installation support when you purchase the product, but must buy a maintenance contract after that time period.
- ☐ If you are still baffled, consider picking the brains of the DBAs who monitor resources, such as the `comp.databases.oracle` Internet newsgroup.

PHASE 4: PROBLEM SOLUTION

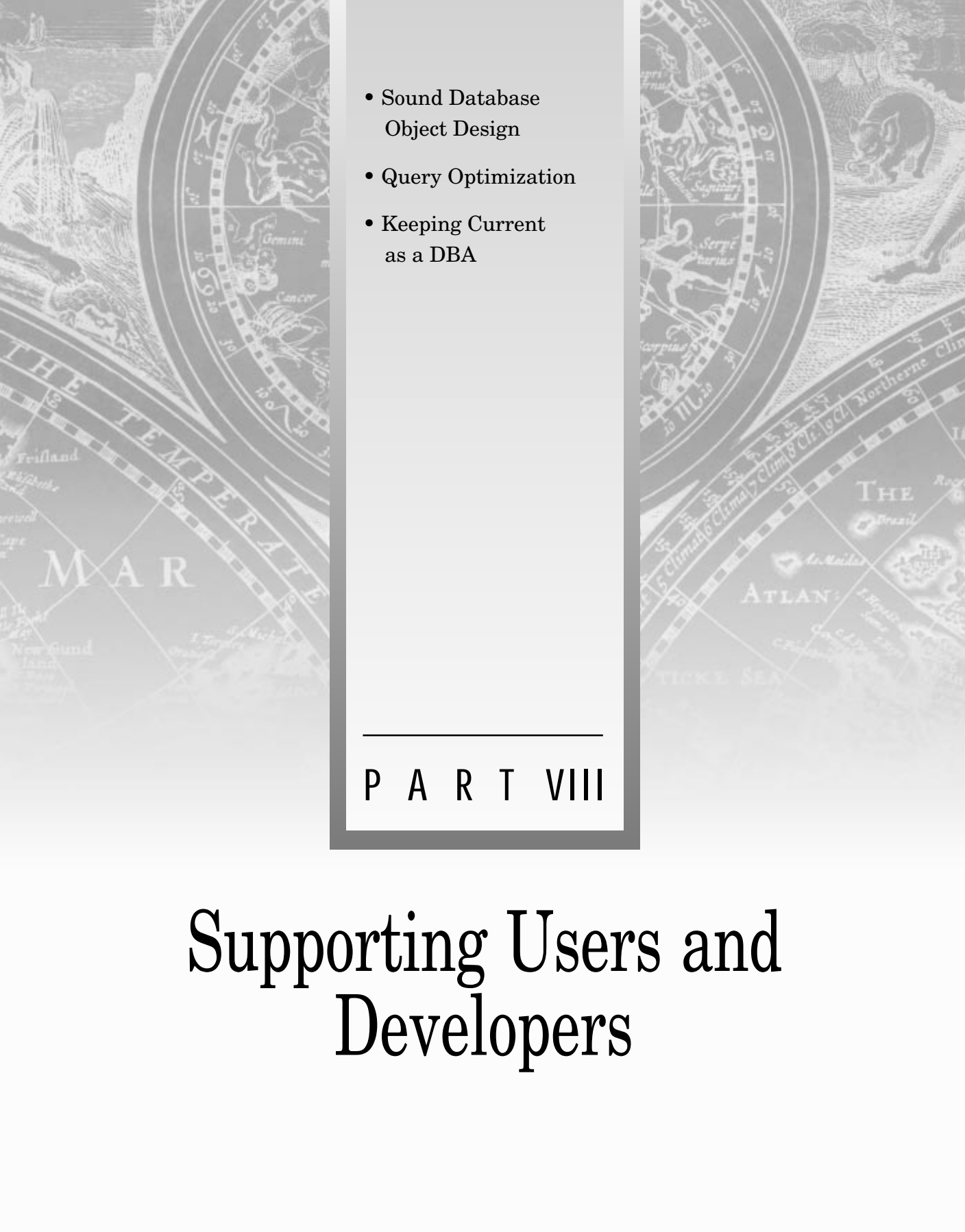
- ☐ Come up with a solution for the problem based on your research.
- ☐ Ask yourself whether the solution has the possibility of causing data loss. If so, consider taking a backup before you try to fix the problem.
- ☐ Ask yourself when you will be permitted to implement your repairs. Will it interfere with normal operations and therefore have to be scheduled after hours? Would you prefer to do it after hours when you can concentrate better?

- ☐ Ask yourself how complex you expect the repair work to be. If it is fairly complex, do you want to write out a quick procedure to follow to ensure that you remember all the details and perform all the steps?
- ☐ Who should you notify about the solution that you are going to implement? Who should you notify when the repair work is completed?

SUMMARY


This chapter provides a quick checklist that you may consider reviewing when you tackle your first major problem with your Oracle instance. You still need to go to the specific manuals to look up your error messages. You may also need to seek outside support to figure out what to do. This checklist should jog your memory when you are in the heat of battle and helping someone solve a problem.

This chapter also concludes the discussion on solving problems that creep up in your Oracle system. You can expect to have some problems even in the best-monitored and best-planned instances. You hope to minimize the problems and have a sound methodology for solving them. You should also spend a little time lining up some of the resources mentioned in these chapters so that you are ready to go when actual problems arise. Here's hoping that you never see a problem, but if you do, you will solve it quickly.

- 
- Sound Database Object Design
 - Query Optimization
 - Keeping Current as a DBA

P A R T V I I I

Supporting Users and Developers

- 
- Overview
 - Normalization and Table Design
 - Table Design Modifications for Decision Support
 - When and How to Use Indexes
 - Naming Conventions
 - Summary Tables Versus Views
 - Sizing Tables
 - Sizing Indexes

CHAPTER 39

Sound Database Object Design

What is *sound* object design? There are many different approaches to software development, many of which can get the job done. The word “sound” implies that the design does not have any errors that could cause you to be unable to retrieve information. This chapter teaches you sound object design and emphasizes how your object design can help you work with Oracle instead of against it.

In Chapter 40, you explore optimization of queries, a task you might be given. If it is not part of your job, that chapter will give you an appreciation of the process so that you can make an informed assessment of whether performance problems result from a need for application tuning. With today’s job market, you might need to have a varied set of skills to help you land your next job.

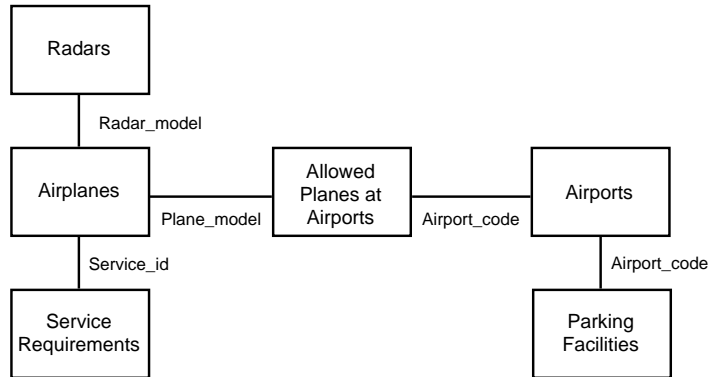
Although this chapter begins a new part in the book called Supporting Users and Developers, it includes a chapter that many might think of as supporting yourself (why not be nice to yourself occasionally). Chapter 41 shows you how to help the users by helping yourself to gain more knowledge and skill. In that chapter, you explore some of the resources that you can tap to keep current as a DBA. Technology is moving very rapidly today. Every indication is that the pace of change will be even greater in the future as Oracle and other database vendors try to move into video on demand, object orientation, and multimedia storage.

OVERVIEW

So far, I have used the word “sound” to describe the nebulous concepts of lacking errors and enabling you to access the information that you need. Coming from more of a scientific and engineering background, I sometimes find it funny to read articles where people get really worked up about one design technique or storage concept that is supposedly the end to all known computer problems (remember CASE tools, reengineering, and executive information systems?). This chapter discusses some of the basics of relational database design, then extends into more detailed topics, such as the use of indexes, naming conventions, and table sizing.

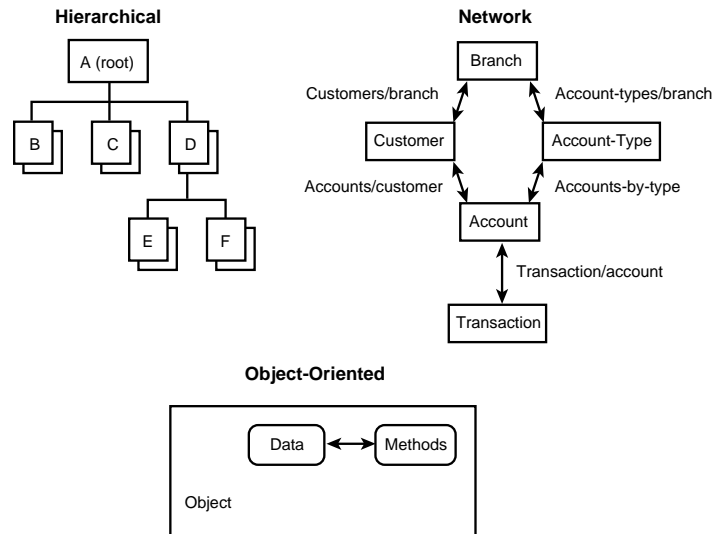
Oracle is basically a relational database management system (see Figure 39.1). Although it has recently gained some constructs that could loosely be construed as object-oriented, for your purposes, consider it to be relational. The key to a relational system is that you have a number of tables that are linked to one another via key fields. There are no explicit pointer columns, and you can have relationships wherein a row in one table maps to multiple rows in other tables. You learn about a few more of the characteristics of the relational model later in this chapter.

Figure 39.1.
Basics of relational
data models.



For now, let's look at some of the other common database technologies just to get a feel for what relational databases are all about. Figure 39.2 shows a few of the more common of these technologies. The mainframe world has a lot of hierarchical databases, such as IMS, which are based on a series of parent-child relationships. The Network model (for example, IDMS or Total) has a series of many-to-many relationships that are related to set membership. Finally, the object-oriented databases mix the data and the actions taken on the data into objects.

Figure 39.2.
Other common data
models.



Most current development efforts are being done using relational database management systems. Examples of relational systems include IBM's DB2, Oracle, Sybase, dBASE, and Informix. There is still a lot of legacy data in the older data models, but that will change (even mainframe shops are tending to move toward DB2 and other systems to improve software development). The relational data model has a few important properties:

- ◆ Each column has a distinct name within a given table and the table has a given number of columns.
- ◆ Each column is homogeneous—each element is of the same data type and is the same kind of data (for example, phone numbers).
- ◆ Duplicate rows are not permitted.
- ◆ Each entry (column value) must be a single data item, not a repeating group.
- ◆ Both rows and columns can be viewed in any sequence and there is no implied meaning to the ordering.
- ◆ All information is explicit data values in some row/column position. No links or pointers are visible to the user connecting one table to another.

There are a few terms that are important when talking about the linkages between tables in the relational model. The first is primary key. A *primary key* is a unique identifier for a row in the table that consists of one or more columns from that table. This is in contrast to a foreign key. A foreign key consists of one or more columns in a table that must match the values of a similar set of columns in another table or be null. In this manner, you can validate one or more columns against a list of legal values that is enforced by the database.

This is a good time for a quick review of the objects that you will normally run into in your database. The basic unit of data storage is the table. Synonyms provide alternate names for the tables or views. Indexes contain one or more columns from the table and are used to provide rapid lookups of specific rows in the table. Finally, views link one or more columns from one or more tables to give the appearance of a single table to queries and transactions.

Now on to something every politician talks about, but rarely achieves—*integrity*. There are two forms of integrity that are applicable to databases. The first is *entity integrity*. This means that no columns in the primary key of the database table are null (and they are, of course, unique). The other form of integrity is *referential integrity*. It applies to tables that have foreign keys that link to primary keys of other tables. To have referential integrity, every value of the foreign key in the first table must either be equal to the value of the primary key in some row of the second table or be null.

Another important concept in relational systems is *data independence*. In Oracle, the definitions of data (column names, data types, sizes, table names, and so forth) are stored in the database itself. The user does not know or care where and in what order the data is actually stored. The user merely specifies what is needed (table and column names), not how to get it. Oracle provides its own internal access methods that are optimized toward rapid data storage and retrieval. (It seems odd. I can remember building algorithms to store data in clever ways in flat files, but I cannot imagine having to do it again.)

It is also important to remember the distinction between the physical and logical design. Recall that you, as the DBA, control the physical layout to some extent. You assign data to a particular tablespace and place the data files on particular disk drives. However, much of the control of where the actual bytes of information reside is inside Oracle itself. Designers need to worry only about the logical design. This consists of focusing on tables, keys, indexes, views, and so forth.

All these neat concepts can be summarized under the term data independence. It is important that you consider these concepts when you are designing (or reviewing the design for) your databases. Data independence provides several key benefits:

- ◆ Productivity is enhanced because the developer does not need to determine the access paths, and SQL data requests are simpler than nonrelational access methods.
- ◆ Flexibility is achieved because changes to the data architectures do not necessarily impact existing application programs.
- ◆ Finally, maintainability is improved because well-designed relational structures tend to be more stable (you do not have to rewrite everything and redo all data files to widen a column just a little).

There are entire books written on relational and other design methodologies. This section highlights those that even a DBA should know. You may not be responsible for optimizing applications, but you should be able to recognize situations wherein the application design may be limiting performance (as opposed to having a problem with your database). Even if you do not work with it every day, tuck this knowledge away. It may come in handy some day.

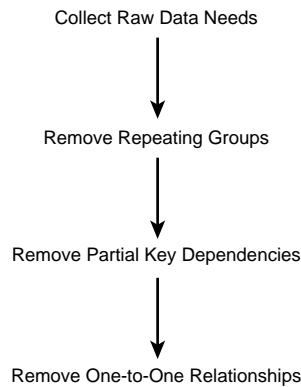
NORMALIZATION AND TABLE DESIGN

Normalization—what a word. Apparently, the people who designed this concept wanted to imply that if you do not follow their rules, your data would be abnormal. This section provides a brief overview of the concepts of normalization—at least enough for your typical DBA. The first thing that you have to do is forget everything about Oracle structures, SQL queries, and specific development tools. All you should

care about is the conceptual data elements that are needed by an application. The *normalization* process takes this data and organizes it in a way that is efficient for relational database processing.

The basic process starts with the data as the end users view it. You then run a series of analyses to put the data into groups that are logically related to one another and represent real world relationships between the data elements. It is an iterative approach wherein you perform one task in each stage. The results of a stage are then used by the next stage to refine the data further. Figure 39.3 illustrates the basics of this process.

Figure 39.3.
Basics of normaliza-
tion.



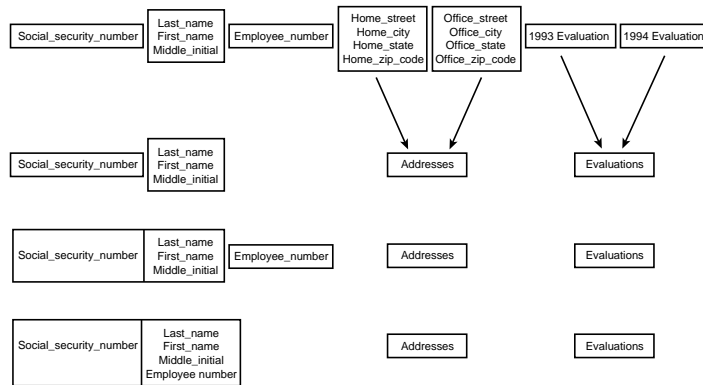
Collecting the data as the users see it is mostly a listening process. You may collect paper forms that are currently being used, data dictionaries from existing computer systems, and other forms of data to speed up this process. In the end, you are left with what is usually a disorganized model of the data (called the *un-normalized form* by some) similar to the following list, which might match up with a simple personnel system:

- ◆ Last name, first name, middle initial
- ◆ Social security number
- ◆ Home street address, city, state, and zip code
- ◆ Summary personnel evaluation score for 1993
- ◆ Summary personnel evaluation score for 1994
- ◆ Office building address, city, state, and phone number(s)
- ◆ Employee number (same as social security number, but with a different name)

Most personnel systems are far more complicated than this, but let's keep the examples for this book somewhat reasonable. The second step in the normalization process is to take the collection of data as the users see it and remove repeating groups. Because everything has to have a fancy name, you are trying to produce the *first normal form*. If you look in the previous example, you see two types of repeating groups. The obvious one is summary personnel evaluation scores (one for 1994 and one for 1993). Another one that might not jump out at you is addresses (one for work and one for home). You can group these together as addresses and put an identifier in to say whether it is home or work.

The fourth step in the process (and the final one that we will go through here) is removing any relationships that are one-to-one. Suppose you start out classifying name, social security number, and employee number as separate data elements. You have to keep employee number as a separate field even though it is the same as social security number, because personnel wants to be able to assign different numbers in the future. However, all three of these elements map to a single individual. Therefore, you can classify social security number as the primary key and the remaining fields as data elements that relate to the individual specified by that social security number. Figure 39.4 illustrates the basic process described in these examples.

Figure 39.4.
Normalization ex-
ample.



39OSD04

There are higher forms of normalization; however, this is a survival guide. If you can get the data structures into this state (you guessed it—it is called the *third normal form*), you are probably doing as well or better than most places out there. If you wish to study normalization in greater detail, there are a number of books on the subject of database design. For now, just have a good feel for the process of taking data scattered about the way the user sees it and turning it into a group of tables that are logically related to one another and well-suited to efficient Oracle database operations.

TABLE DESIGN MODIFICATIONS FOR DECISION SUPPORT

Wasn't that a good theoretical lecture about how to make things nice for Oracle? You construct a number of tables and divide everything into neat buckets. The only difficulty is that when you want to print out a complete personnel report, you have to issue queries against a number of tables and use the `where` clause to join all the tables. That is fine and worth the effort for professional programmers. They should be quite comfortable with (or at least tolerant of) that type of work. What do you do in the decision support arena wherein you may be giving those easy access query tools to your end users?

Odds are that your average end users will not be able to be productive in an environment with such a large number of tables. If you have an instance that contains a wide range of data, they may have trouble finding all the tables that contain the information for which they are searching. Therefore, you may have to de-normalize your table structure a little to accommodate the decision support users. There are two ways to approach this: de-normalizing your table structures or creating views that fuse the data for the users to make it look as though you de-normalized your tables.

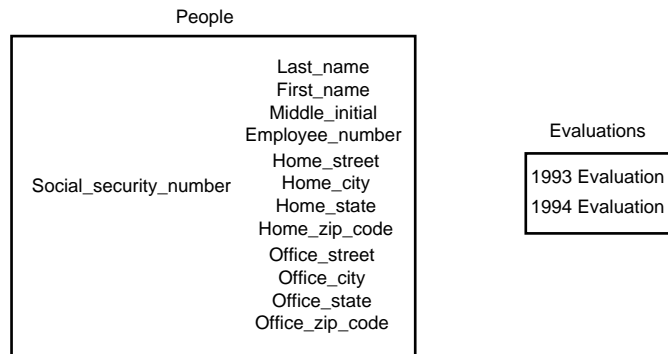
Let's investigate the first option. It is handy to separate things into nice conceptual buckets (people, addresses, evaluations, and so forth). It generally makes for more efficient data storage and can even improve response time because you are not reading through a lot of information that the user does not typically need. However, you want to make it easy for the decision support users, so you might consider putting the home and work address information in the basic personnel record. You will be limiting yourself somewhat (you cannot accommodate people with multiple office addresses, for example). However, it will make it easy for the users to form their queries—they have to deal with only a people table and an evaluations table (see Figure 39.5).

Perhaps you do not want to do this. Your system may get fairly large and the inefficiencies and inflexibilities introduced by such table structures may be too much for you. You do have an alternative. You can create views that fuse the people and address tables together. When you publish your schema to the decision support users, you present it as though it looked like Figure 39.5. Only you will know that they are accessing the people view that accesses the people and address tables.

Enough on this subject. I just wanted to let those of you who have the pleasure of working on decision support get a feel for the de-normalization process. It may not be as efficient for the computer (although it can save a lot of data transfer time if you typically retrieve only a few rows from larger tables due to Oracle's system of

retrieving whole blocks of data for all queries). However, it may make the difference between end users being able to build ad hoc queries or not. If you do not like messing with your tables, you have the option of creating views in which you store the SQL that is needed to join the tables properly and then enable the users to hit against these logically simple structures.

Figure 39.5.
De-normalized table
structure for decision
support.



WHEN AND HOW TO USE INDEXES

Indexes can make a world of difference to your queries. I have seen queries that ran for five minutes without an index take less than thirty seconds when the indexes are put in place. The basic principle of indexes is quite simple. If you routinely issue queries against a special key within the table (for example, social security number in a personnel system), you create an index that contains only those key values. These values are ordered within the index, and Oracle uses a highly efficient algorithm to search the index. The end result is that you get much better response time. It also results in a system that performs much better even as you increase the size of your tables.

With all those glowing words about indexes, there have to be some negatives, too. One of the biggest disadvantages for people who are tight on space is that it takes disk space to store these indexes. You also have to keep the indexes (all of them) up to date when you add data to the table; therefore, your data insertion, update, and deletion routines will take a little bit longer to complete. You have to consider how much of your system's time is consumed actually updating the database and how much time is spent running queries.

Another important consideration about indexes is that they are useful only when you are issuing queries on the fields contained in that index. This means that if you search against fields that are not in the index, it will not be used and you will wind up doing a full table scan. Examples of tables that work well with indexes are

reference tables wherein you are routinely looking up a translation for a value entered by a user or all of the items in the table that match a given set of input criteria. If you are not sure about the overall effect of a particular index on the overall efficiency of an application, consider trying out the index (preferably in a test instance). If it does not work out, you can get rid of the index with a single `drop index` command.

The next question that often comes up is how many indexes to construct. What about the situation where you normally query on last name, but occasionally go on both last name and first name? Oracle's indexes can accommodate these needs with a single index (last name and first name). Oracle will detect when you try to create two identical indexes, but will not if you try to create indexes that overlap one another. Therefore, always look for ways that you can have one index serve multiple purposes. It helps you both by minimizing the impact on updates and by saving disk space.

Finally, you have to be sensitive about the order in which you query for items in Oracle. If you list your criteria in an order that does not match the order of the criteria in the index, the execution plan may not be smart enough to figure it out. The newer optimizers (version 7.1 and later) are usually smart enough to handle this. The older versions were not that smart and used the order of items in the `WHERE` clause to determine which indexes to use (I know the Server Concepts Manual said that it did not matter, but we proved that it did). However, it is not that hard to put your criteria in the order of the index and it may help jog your memory about trying to use indexes.

There are cases wherein you get an index without explicitly asking for it. That may seem strange, but it is true. You just have to know when you are indirectly asking Oracle to create an index. There are two common cases wherein Oracle needs to create an index to do its job. The first is when you define a field or a set of fields as a primary key. The primary key status implies that you have to have unique values for the field or combination of fields that make up the primary key and that none of the keys are null. The other example of an implied key is when you designate that a field or combination of fields have a unique constraint placed on them. To ensure this, Oracle creates an index to keep a sorted list of the keys. When you insert a value, it checks this index to see whether you have already entered these key values. If you have a duplicate, the insertion (or update) will fail and you get a message telling you that you tried to violate the primary key.

Where are all these implied indexes created? If you do not specify storage parameters, they are created in the default tablespace and sized according to the sizing defaults. They are given names such as `SYSnnnnnn`, where `nnnnnn` is a unique

number in your system. If you balance input/output loading by placing tables on separate disks from their indexes (as you learned from the previous discussion), this situation can be a problem. Therefore, you should be familiar with the syntax of the SQL `create table` command that allows you to control the creation of these indexes:

```
create table table-name
(field-list
...,
constraint constraint/index-name
primary key (ssn)
using index tablespace tablespace-name
storage (initial #
next #
pctincrease #
minextents #
maxextents #))
tablespace table's tablespace-name
storage (initial #
next #
pctincrease #
minextents #
maxextents #)
```

Here is an example of creating a table with the storage of the index specified:

```
SQL> list
1  create table personnel
2  (ssn          varchar2(9),
3  first_name    varchar2(20),
4  last_name     varchar2(30),
5  middle_init   varchar2(1),
6  constraint pk_personnel
7  primary key (ssn)
8  using index tablespace user_data
9  storage (initial 8K
10 next 4K
11 pctincrease 0
12 minextents 1
13 maxextents 100))
14 tablespace user_data
15 storage (initial 20K
16 next 4K
17 pctincrease 0
18 minextents 1
19* maxextents 100)
SQL> /
```

Table created.

SQL>

Now that you have explored how to create *implied* indexes in some detail, it's time to look at the format for *explicitly* creating an index in Oracle. This command goes something like the following:


```
create index index-name
on table-name (column-list)
tablespace tablespace-name
storage (initial #
next #
pctincrease #
minextents #
maxextents #);
```

Here is an example of an index using the personnel table:

```
SQL> create index personnel_names
2  on personnel (last_name,first_name)
3  tablespace user_data
4  storage (initial 4K
5  next 4K
6  pctincrease 0
7  minextents 1
8  maxextents 100);
```

Index created.

SQL>

So, back to the original question—when do you use indexes? If you are using a data warehouse, you are probably doing mostly queries and therefore you will benefit from constructing indexes for all common queries that have significant response time (forget the 10-row minor lookup tables). If you are a transaction processing system, you should typically index the main data tables on their primary key and index the reference tables on their most common lookup keys. Other than that, keep indexes to a minimum, but watch how your performance goes. Try an index for those really nasty queries.

NAMING CONVENTIONS

Perhaps you have noticed that this book stresses organization, patterns, and rules when approaching database administration. The subject of naming conventions gives me the opportunity to get up on my soapbox once more and preach the gospel of organization. Don't get me wrong—I have worked under a lot of different naming conventions that have worked equally well. I am not going to specify a naming scheme that is the only one any intelligent person would ever follow. Rather, let's look at some of the criteria of a good naming scheme.

First, it is important to establish the purpose of naming conventions. Remember the days when programming languages enabled you to have only variables such as X1 or B2? (If you don't, just trust me on this.) A few weeks after you wrote the program you could not figure out what was going on without looking at the documentation

that you wrote—if you wrote it. Then longer variable names came onto the scene and you were able to create variable names such as `last_name`. Suddenly, other people could look at your code and figure out what was going on (unless you were one of those nasty folks who wrote self-modifying code or used a lot of pointers). This accomplished the purpose of naming conventions—enabling people to look at the name of a table or a field and know what you put there.

Naming schemes that are easy to follow become important in two particular situations. The first involves systems wherein there are a number of programmers working on many tables. It can really slow things down if you have to look up fields in a book every time you write a query. The other situation in which naming schemes that are easy to follow are important is in decision support systems where end users are given query tools. It can really help reduce the load on information systems to code all the reports that the users are creating. However, most of these tools rely on good names for database objects and columns to make them useful and productive.

So what are the criteria for naming conventions? You may have some local traditions that everyone knows. Those will be useful and may have an impact on some of your decisions. However, the following list is a good start at criteria for a naming convention:

- ◆ It has to be simple enough for people to follow.
- ◆ You should be able to distinguish between tables, indexes, and views (and perhaps stored procedures if you use them).
- ◆ The name of the object should reasonably convey what is being stored in it to both other developers and end users.
- ◆ You should keep your object names less than 30 characters (and you cannot use blank spaces, so use underscores or dashes).
- ◆ The name of an index should indicate the table on which it is based. You can either number indexes sequentially (1, 2, and so forth) or try to fit a description of the fields that they cover within your 30-character limit.

Here is an example of a system that is like many in operation:

- ◆ All tables begin with `t_`.
- ◆ All indexes begin with `i_`.
- ◆ All views begin with `v_`.
- ◆ All stored procedures begin with `p_`.
- ◆ Some examples might be: `t_bills_sent`, `v_people`, and `i_addresses_zip_code`.

SUMMARY TABLES VERSUS VIEWS

If I offered you improved response time in return for some disk space, would you take it? Some people, tight on disk space, would say that response time is good enough as is. Others—probably a lot of the data warehouse folks—would probably jump at the change to save 25–50 percent on some of those really nasty queries. That is what summary calculation tables offer you. If you find that your users frequently issue queries that require the calculation of a common set of sums, averages, and so forth, you can calculate those sums in advance and enable the users to access the summarized information through simple queries.

This is actually quite simple to implement. Look at the SQL that is being used in your queries. If the summations are always taken across the same entity (for example, summarized by store), you can use that SQL to build a table that is grouped by that entity. You can run these summary calculations during off hours and save your database a lot of processing time during the busy daytime hours. The tradeoff comes when you have to allocate disk space for these summary tables and their indexes.

An alternative is to make the SQL easier for the users without avoiding the computation in real time by creating a view that fuses the tables or calculates summations. When you issue a query against the view (which looks like a simple table to the users), Oracle still goes through the calculations, joins tables, and so forth. You may get some speed improvement because you can optimize the SQL for the view and avoid having to debug a large number of programs that may have poorly written SQL. The advantage of a view is that it looks simple to the end users (they access columns that might be called `summ_costs`, which are actually a complex series of hundreds of rows of data), and you can control the SQL. It can be another useful tool to keep in your arsenal for certain situations.

SIZING TABLES

An important part of sound object design, which goes beyond laying out the fields into a good series of tables, is sizing the tables and other objects correctly. As mentioned in some of the performance-oriented chapters, if you scatter a large number of extents across multiple disk drives and cylinders, your overall system performance will suffer. This section presents three levels of detail for sizing your tables. These methods differ from one another in the amount of information that is required to begin the process and the amount of time it takes to work through the calculations.

The first method for sizing a table is the rough guess method. This is often used when you have very little information about the table or system (for example, development doesn't start for another three months). In this method, you begin with a system or table that you know the size of and make an analogy between the new system or table and the old one. Examples might be "about the same size" or "half as big." It's not very precise, but sometimes you just have to take your best shot at it.

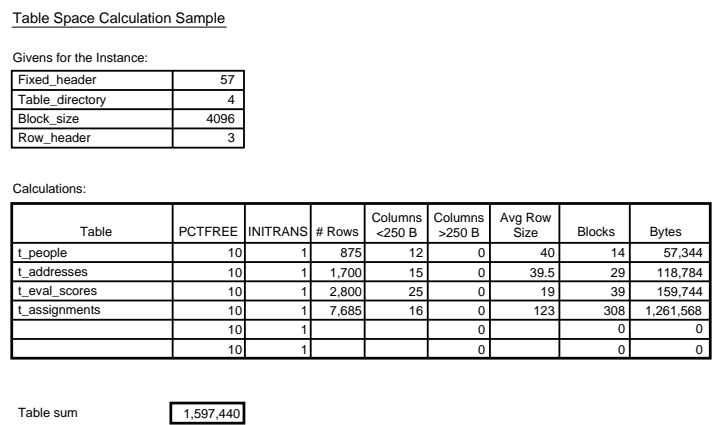
The next method is somewhat more precise: sizing up based on current sizing. If you have an initial production set of data or perhaps a reasonably sized test instance, you can make an analogy between the current size in bytes and the number of current rows to the expected size in bytes and the expected number of rows. This is a sound method; however, it is complicated by the fact that Oracle sometimes allocates space in large extents. You do not know whether the extent is almost empty or almost entirely full. This can cause a wide variation in the accuracy of your calculations. If the table or index has become highly fragmented, this may not be a problem. For example, if you have 100 1M extents, you have between 99M (the last extent is empty) to 100M (the last extent is full), which is only a one percent error. (If we could always be that accurate!) This is much quicker than the next method, and if you have some good data under your belt (for example, a test instance that ran into multiple extents), you are probably in decent shape.

The final method is the most accurate: the row sizing method. It is based on your querying the database to find the average size of a row of data on a table-by-table, index-by-index basis. To do this, calculate the estimated number of rows (number of people in the database, number of orders in a seven-year period, and so forth). Then measure the average size per row with a query similar to the following:

```
select avg(nvl(vsize(ssn),0))+  
avg(nvl(vsize(last_name),0))+  
avg(nvl(vsize(first_name),0))+  
avg(nvl(vsize(middle_init),0))  
"Space of Average Row"  
from personnel.people;
```

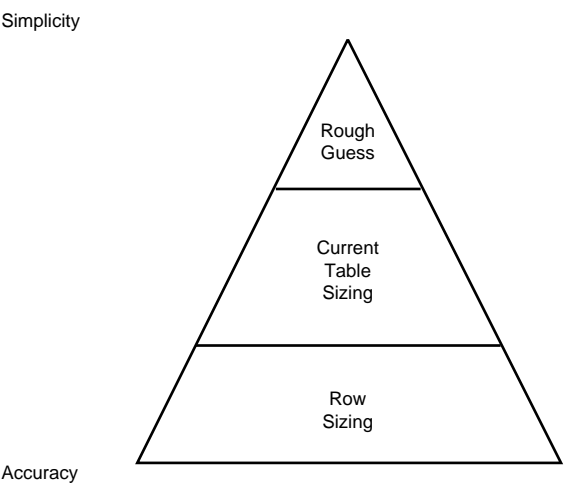
Once you have these numbers, you put them through a relatively complex series of calculations based on Oracle storage methods and come up with an estimated total size for the table. It is important to remember that it always gets allocated in whole blocks. When you have a little 200-byte table with a 4096-byte block size, you will consume at least 4096 blocks when you create that table. The details of this calculation can be found in the Server Administrator Guide. The enclosed disk has a spreadsheet which has been created from this procedure. It has a lot of hidden columns (full of preliminary calculations that you do not need to see) and can really save a lot of time with calculations. The spreadsheet is called "sizer." Figure 39.6 shows an example of the output of this spreadsheet.

Figure 39.6.
Sample table size
calculation spread-
sheet.



Now, of course, you can back off of this method to perform the nauseatingly accurate calculations only on your major tables. Who cares whether you make a 50 percent error on that 1000-byte lookup table? You could use one of the less-accurate methods for the little guys. The three basic methods for calculating the size of the tables are shown in Figure 39.7. You can pick the one that best reflects your level of knowledge of the table and the amount of accuracy that you need (and possibly the amount of time that you have to perform these calculations).

Figure 39.7.
Table sizing methods.



Whichever method you choose, your goal now is to use the output of your work to size the tables correctly that you create. This will be reflected in the initial extent and

next extent parameters in the storage clause of the `create table` command. Try to get everything into a single extent and make the next extents about 10 percent of the size of the initial extent. That way, you will get warnings in your utilization report that tables are starting to fragment well before they start to exceed their extent limits. It is a sign that perhaps something is happening that can urge you to recalculate some of your storage needs.

SIZING INDEXES

The other major data storage mechanism that eats up space on your system is the index. You may find that you have a large number of indexes and they can take up a substantial amount of space on some systems, especially those where query performance is important. The good news is that you have the same three methods available to you that you used for tables. Only the last method requires some updating for indexes (and it is not really that much of a change).

The first change that you have to make in the row sizing method for indexes is the columns against which you calculate the average size. You have to calculate the size of the columns in the table, but count only those columns that are part of the index. It is really much shorter and easier to write the SQL queries to determine the average row size of indexes than tables. Most indexes have only a couple of columns.

The other change involves the calculations that you go through. Again, the Server Administrator Guide explains the details of this calculation. This is a second page on the sizer spreadsheet, which is on the disk for this book. Once again, you plug the numbers into the blanks on the spreadsheet and it calculates the results for you. Figure 39.8 shows some sample calculations for indexes.

Figure 39.8.
Sample index size
calculation spread-
sheet.

Space for Indexes Calculation Sample

Givens for Instance:

Fixed_header	113
Block_size	4096
Entry_header	2
Rowid_length	6

Calculations:

Index	INITRANS	PCTFREE	# Rows	Columns <250 B	Columns >250 B	Est. Avg Data Space	Blocks	Bytes
t_people	2	10	875	2	0	15	7	28,672
t_addresses	2	10	1,700	3	0	12	12	49,152
t_eval_scores	2	10	2,800	2	0	8	15	61,440
t_assignments	2	10	7,685	4	0	18	69	282,624
	2	10			0		0	0

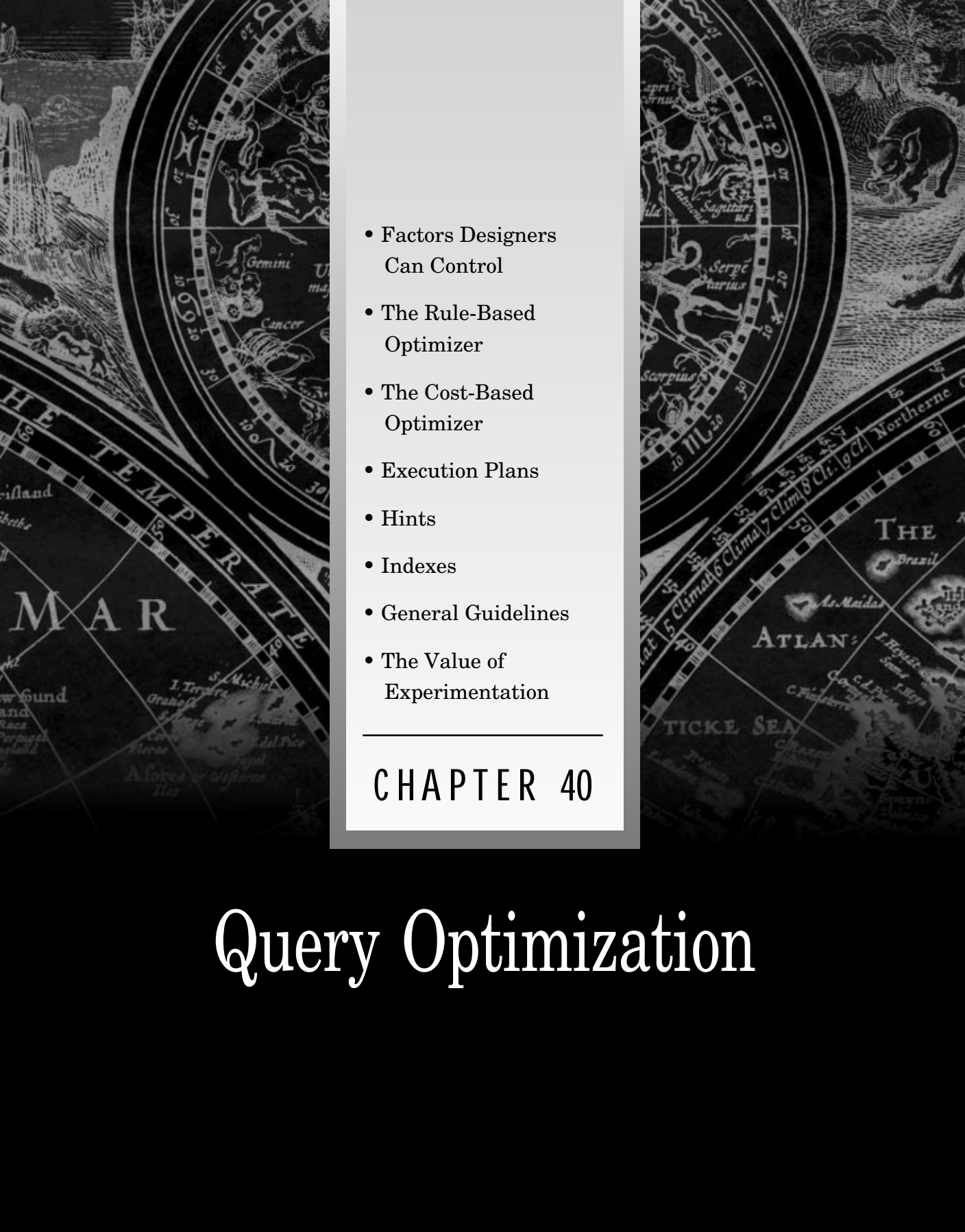
Index sum

421,888

SUMMARY

This chapter tackled a somewhat broad subject. It may be a difficult one for DBAs who have not done a lot of software development or data modeling—it is a skill set unto itself. However, you should at least be familiar with the concepts of sound object design. This will enable you to look critically at your instances to see whether their performance may be limited by the object design.

The next chapter takes some of these ideas a little further. Once you have sound object design, you need to ensure that you interact with the sound database in an efficient manner. Typically, modifications to the database tend to be relatively straightforward (add a new address for this person, for example). The real performance impacts tend to come from the efficiency of the query designs, and Chapter 40 provides brief insights into this process. Who knows—perhaps you can look like a hero by tweaking the order of fields or inserting a hint into a query that causes a 30-minute query to run in less than a minute.

- 
- Factors Designers Can Control
 - The Rule-Based Optimizer
 - The Cost-Based Optimizer
 - Execution Plans
 - Hints
 - Indexes
 - General Guidelines
 - The Value of Experimentation
-

CHAPTER 40

Query Optimization

“It’s not my job.” Perhaps tuning applications is not your problem. Perhaps you already have more than enough problems to deal with and do not want to get into any more. However, you still should have a basic understanding of application and query tuning. Why? Because odds are, if you have a performance problem with the system as a whole, the first words out of people’s mouths are going to be about the need for database tuning—they will tell you about how messed up your system is. You should, of course, perform the database tuning steps listed in the previous chapters to ensure that you know what you are talking about before you open *your* mouth. However, once you prove that the database is performing within normal limits, you may want to be able to track down the real cause to make your case more convincing. That is where a little knowledge about application tuning can make you dangerous.

The majority of application tuning work that I have run across involves optimization of queries against the database. Although you may think up situations where you might want to tune transactions, most of these situations lead to splitting input/output loads between disks and other such things that are grouped within the realm of database tuning. However, generally speaking, the process of updating rows, deleting rows, and inserting new rows is relatively well-defined and follows a single course. The most that you can do is make sure that you have indexes that match the typical insertion criteria used and let Oracle do its work.

Queries, however, are another matter. The real value of database management systems to most organizations is not the capability of storing data (flat files did that successfully). The real value of a DBMS is the capability of rapidly and accurately recalling data from the database. This is where man-years of labor were spent in older systems as each developer had to build smarter query algorithms to find that data in a time frame that was reasonable. Not every query is the same though. Some may require you to scan the entire table (that is, you use almost every column in the table as part of your `where` clause). Others may look only at a single, indexed column in a single table. Still others may link a half-dozen tables using complex join criteria.

It seems pretty obvious that a search method that works well for one of these types of queries is probably not the best method for the others. In fact, Oracle and most other databases have a number of search algorithms in their systems that are designed to handle different needs. This is what enables them to work in such a large number of environments and applications. When an application is performing slowly on a well-tuned database, the first thing you should check is how the query is being processed to ensure that it is as efficient as it can be.

This chapter explains what you can do to make a query run more efficiently in an Oracle system, starting with a discussion of the factors that you can control when issuing a query. Then you learn about the two optimizers that Oracle provides. These are the components within the system that determine the search algorithm,

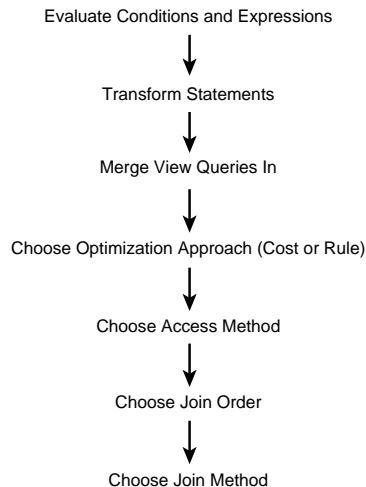
or execution plan, that will be used to process the query. Then you explore steps that you can take to optimize the query. Finally, being trained as a scientist, I go over a brief discussion on the value of experimenting with different factors to optimize the query. Unless you devote a percentage of your time to this discipline, you will probably not be able to look at a query, make two changes, and then watch as it flies.

The level of detail presented in this chapter is definitely at the survival guide level. There are books devoted to this subject. There are a number of tricks and methodologies that can improve software dramatically and a lot more that could be covered about how the optimizers and execution plans work. However, most DBAs do not have to do a lot with query optimization, so this chapter presents just enough information to get you going.

FACTORS DESIGNERS CAN CONTROL

The process begins with a request for information. In an Oracle database system, this usually starts with a SQL query. Some development tools give you a graphical interface to enable you to tell it what you want, but these are merely convenience tools that eventually generate the SQL query and pass it to Oracle. Once the query is formed, Oracle goes through a number of steps to get the answer to the user. The steps related to the choice of an execution plan and execution of the query are shown in Figure 40.1.

Figure 40.1.
Query execution steps.



The first step in figuring out ways to make queries perform better is to understand the factors that you can control. Perhaps one of the most frustrating things about query optimization is that the basics of SQL do not provide you with much in the

way of direct controls over the query execution process. Of course, this lack of control is one of the things that people like most about a database management system—it handles all the details. In most cases, the DBMS will choose a plan that is far better than most programmers can implement. However, especially in large databases with large tables, you may need a little bit better control in those cases where Oracle chooses a plan that does not work out well.

The first item that you need to control in Oracle version 7.1 and later is which type of query optimization Oracle performs. There are two optimizers available to you in version 7 of Oracle. The first is the latest version of the optimizer that Oracle has used for some time—the *rule-based optimizer*. It gets its name from the fact that Oracle has programmed in a series of rules about how to read the SQL statement in order to determine the execution plan. You can write some really poor queries if you are not careful with this optimizer. The other optimizer is referred to as the *cost-based optimizer*. It looks beyond the pure format of the SQL that you write to determine the sizes of the tables and indexes involved with the query to then determine the best execution plan. Even though this option is available in Oracle 7.0, no one that I know of ever got it to work correctly. I had fairly good results when I tested the cost-based optimizer under 7.1, although there were still a few queries that I can get better results with by using manual tuning.

Once you choose your optimization method, you have to work with your queries to help ensure that the optimizer chooses the best method for your needs. If you are using the cost-based optimizer, you need to ensure that you routinely take statistics on your database objects. This ensures that Oracle has the information that it needs to make wise decisions. This optimizer ignores the order in which you list items in your SQL statements; therefore, your only choice is to override the optimizer (as described later in this chapter).

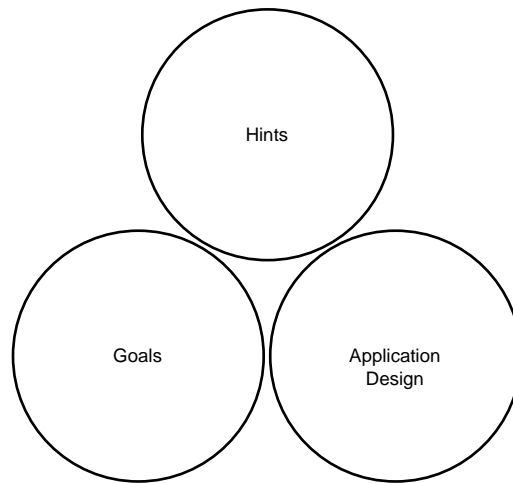
If you are using rule-based optimization, you may have some options for using the layout of your SQL statement to ensure that Oracle picks a good execution plan. The book for Oracle 7.0.x says that the order in which you list items in your SQL statements does not affect the execution plan chosen. I disagree. I have seen some statements where you rearrange two columns and suddenly it chooses a good execution plan and runs like lightning. You can't believe everything that you read. However, I did test it out under Oracle 7.1 and found that the book was correct, because the order of the items in the SQL did not affect the execution plan under either the rule- or cost-based optimizers.

The Oracle optimizers are a nice idea overall. You get to access your data without concerning yourself with search algorithms or physical storage parameters. You do not even have to decide when to use an index or which index to use. This all works well in the vast majority of queries that I have run across. Every now and then,

however, you will run across a query that joins a lot of big tables and it takes forever to execute. This is the time for manual intervention into the execution plan process. Intervention is needed usually only in very large instances (that is, greater than 25G in size).

The forms of intervention in the execution plan process can be separated into three general areas (see Figure 40.2). The first form of intervention is the *hint*. You can add a hint to any of your SQL query statements. You can literally tell it the execution plan that it should be using. You can also tell it the order in which you want the tables to be processed. This can be pretty powerful, but you should avoid using it unless you have queries that have problems. Later versions of Oracle may have even better execution plans available to them. If you code in a lot of hints, you will miss out on the opportunity to use these new execution plans. (Most places do not have the kind of time to go through a large application and clean out all the hints and then retest it for speed.)

Figure 40.2.
Intervention with
Oracle execution plans.



The second form of intervention is *goals*. This construct enables you to tell Oracle whether you wish to choose an execution plan that gets the entire query completed quickly or returns the first few rows of data quickly. For large reports that are not printed until you have completed them, you are typically concerned about getting all rows returned as quickly as possible. However, if you are displaying a list of values on the screen for the users, you may wish to get them a few rows to read while the rest of the query is being completed. This enables them to read and analyze data and avoid screen dead time that leads to user complaints. These goal options use cost-based optimization whether there are statistics present or not. (With no statistics, the chance of making poor decisions is higher.)

The final form of intervention is the application design itself. You can often improve performance by using SQL over PL/SQL constructs. You also can eliminate unnecessary information from the query that may cause additional work for Oracle. Finally, you can try various combinations of query constructs, such as `UNION` and `UNION ALL` to try to improve performance. There are certain constructs that force certain execution plans (such as the dreaded merge-join) that are less efficient. I usually wind up doing a little experimentation with these options to find the construct that works best for a given query.

THE RULE-BASED OPTIMIZER

The concept behind the rule-based optimizer is simple. It looks at the SQL statement to determine all the possible ways that it can execute the query. Then it selects the one on its list of queries that it thinks will be most effective. The key here is that it has a simple list of possible ways to approach the problem that is fixed for all types and sizes of tables. The following is the basic order in which the execution plans are selected by the rule-based optimizer:

1. Access a single row by using its internal row ID.
2. Access a single row by a join formed out of a cluster of tables.
3. Access a single row by combining a hash cluster key with a unique or primary key from a table not in the cluster.
4. Access a single row by a unique or primary key.
5. Access by joining rows in a cluster together.
6. Access by hash cluster keys.
7. Access by using a cluster index key.
8. Access by using a composite index.
9. Access by using a single-column index.
10. Access by using a bounded range search on indexed columns (for example, `>25` but `<50`).
11. Access by using an unbounded range search on indexed columns (for example, `>5`).
12. Access by joining tables using a sort-then-merge algorithm.
13. Access by finding the maximum or minimum of an indexed column.
14. Access by using the `ORDER BY` clause on indexed columns.
15. Access by performing a full table scan (read every row in the table).

As you can see, unless you use clusters, you have a very simple list of ways to attack the problem. It also does not have a clue as to which table to access (that is, which

will return the fewest number of rows) first based on this data. Another point to note is that you wind up doing a full table scan if you do not have a `where` clause to specify rules for joining tables or selecting rows from the tables. It is relatively simple in the way it approaches things and can be easily fooled.

There is a little more to the evaluation process than has been described so far. The rule-based optimizer will catch a few other ways to make the statement more efficient than you coded it, but not many. Here is the list of steps that the optimizers go through:

1. Evaluate your SQL to determine the expressions and conditions.
2. Modify the statements into an equivalent join statement that may be more efficient.
3. Merge the queries used for any views that are accessed by the query into the query.
4. Choose whether to use the rule-based or cost-based approach.
5. Choose access paths for each of the tables accessed in the query.
6. If two or more tables are joined, choose the order in which the tables are joined.
7. If tables are joined, choose the method of joining.

With all of that said, I tend to use the rule-based optimizer unless I have a reason not to use it. First, some of my feelings come from the fact that the cost-based optimizer did not work very well until recent releases of the Oracle product (which is probably an unfair bias). In addition, even when I tried the cost-based optimizer, I found a few statements that I could do better than the optimizer. The rule-based optimizer also has the beauty of being simpler to maintain. I do not have to gather statistics and keep them current. The rule-based optimizer is good enough for the vast majority of databases with which I have worked. There are however, a growing number of large databases where efficiency is essential (you can be inefficient on a 50-row table, but not on a 5-million-row table). These are the cases in which the cost-based optimizer can help you out. There is no harm in trying out the cost-based optimizer in your test environment to see whether it can benefit your applications. Your situation might benefit enormously from this tool.

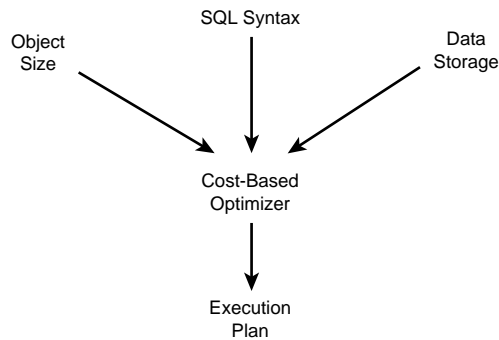
THE COST-BASED OPTIMIZER

Imagine having to design an execution plan manually just based on the SQL for an application and that you know nothing about where the table and column names are written in a foreign language. You have no way to guess which is the smaller, lookup table and which is the monstrous main data table. You have no idea whether you are

dealing with a huge or small query, whether you can perform your sorts in memory, or whether you will have to use disk sorting. How can you be expected to come up with a good execution plan?

These are the problems faced by the rule-based optimizer. It gets only the syntax and a list of objects that it does not understand. It does its best, but it winds up making assumptions that cause it to pick the best plan for the majority of queries of this type. The cost-based optimizer takes a little extra information that you have been kind enough to gather for it and uses this information to make better guesses at the optimal query path (see Figure 40.3). This can prevent, for example, scanning through a huge table to find rows to join to a little lookup table (it is much faster the other way around).

Figure 40.3.
Cost-based optimization.



The statistics that you gather cover several different topics that can be used to determine the most efficient query. It looks at the number of rows and how the objects are stored. Next, it performs some calculations for the impacts on the three major system resources—CPU, memory, and input/output. It then is run through an algorithm (which factors in the goals that you have set up) to determine which query will give you the best performance. I get a headache just thinking about all the calculations that I would have to go through to determine which approach is best for a complex query joining multiple tables. However, that is what the software gets paid to do.

There is probably an entire series of papers that have been written on the internal functioning of the cost-based optimizer. However, this is a survival guide. It is important that you understand the basics of what this optimizer does and what it can do for you. You should seriously consider using it in very large instances, where there is a mixture of small lookup tables and huge data tables. It could be a useful option in any instance where queries are taking longer than desired under the rule-based optimizer. Of course, you always want to ensure that you are properly tuned before you tell the developers to rewrite their applications or before you change optimizers.

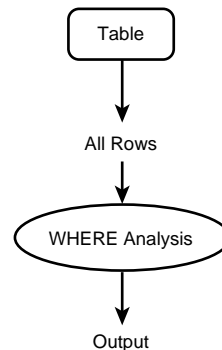
EXECUTION PLANS

Before I wrote this book, I knew basically what the various execution plans were doing and which ones were really bad (merge-join is especially difficult in large databases). However, I had never really taken the time to study the various alternatives. In many cases you, the DBA, will be the resource that people come to for knowledge of what makes the database work more efficiently. It is useful to keep this information tucked away just in case someone asks you what the merge-join means on their explain plans (which is explored later in this chapter).

In order to keep this simple, this chapter focuses only on those items that are not related to clusters (because most clustering that I have run across is done internally by Oracle). You will also learn about the various explain plans, using a mostly graphical format so that it is easy to see the differences between the plans. Key performance highlights of each plan will be explored.

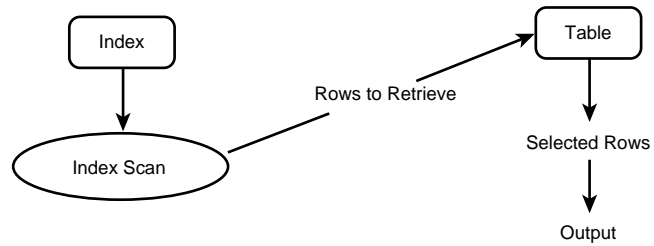
Let's start with the brute force approach to finding information—the full table scan. This basic approach dates back to the days of flat files. The mind-set is that if you read every row in the table, you will find your answer (see Figure 40.4). Although it may not be the most brilliant plan possible, it has its uses. For example, if you have a specific query that is looking for a specific combination of a large number of columns or you want to produce a detailed report showing all data stored in the database, this is exactly what you want. However, if you are looking up a single row of information using a common key value, you will waste a lot of time with this access method.

Figure 40.4.
The full table scan.



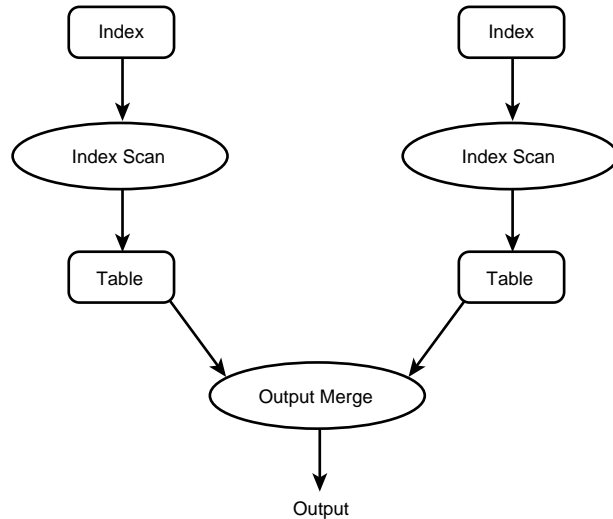
The next simplest method is using an index to access only the rows of data that you need. This typically cuts down on a lot of input/output to the table and also speeds access due to the more efficient storage algorithms used for indexes. Figure 40.5 illustrates the basic concepts of this approach. Obviously, if a table is really small or you are using all the columns of the table in your `where` clause, it is not worth the effort of using an index (because you will read in all the columns of the table when reading the index and then read the table for any remaining columns).

Figure 40.5.
Indexed searches.



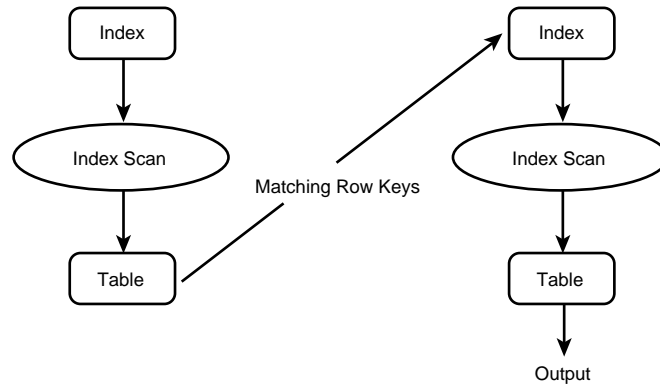
The first two methods deal with access to a single table. These basic table search methods are often combined to obtain data from multiple tables and join it. There are two common algorithms for merging this data—merge-join and nested loop. Let's start with the merge-join algorithm (see Figure 40.6).

Figure 40.6.
The merge-join
approach.



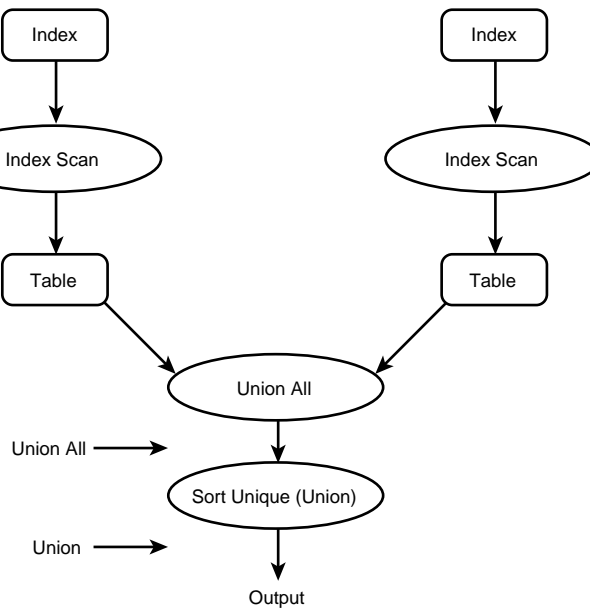
The main alternative to the merge-join is the nested loop. In the large, multi-table queries that you often see with a large data warehouse, you almost always want to see nested loop joins being performed rather than the merge-joins. Recall from earlier in this chapter, the 30-minute queries that turned into 30-second queries. These were queries that initially used the merge-join algorithm and were converted to use the nested loop approach. The basic improvement comes from using the results of the first query to narrow the search of the second, as opposed to performing two independent queries and then trying to mesh the results together. Figure 40.7 shows the nested loop approach.

Figure 40.7.
The nested loop
approach.



There are a number of other algorithms that could be called into play when using compound queries (UNION, UNION ALL, and INTERSECT). Depending on the details of your individual query, some of these may improve your performance. You may have to adjust the SQL in your query to get Oracle to use these statements, but I have seen examples where it is definitely worth it. Figure 40.8 illustrates the concepts behind these algorithms.

Figure 40.8.
Compound query
approaches.



There is a lot more to the process of optimizing queries than can be covered in these pages, and there are many books on the subject. Some good information can be gathered from your server concepts manual and the applications developers guide that normally comes with the complete Oracle documentation set. This is a complex topic with which at least some of your more senior developers should be familiar. Perhaps you may want to loan them copies of these books if you are still having performance problems after you finish tuning your instance (it could save you some time).

HINTS

Some people out there just can't take a hint. Oracle, however, will almost always listen to your hints whenever it can follow them. Hints are your primary weapon when you have a query that is basically sound and exactly as you want it to be, but Oracle still is not processing it the way you want. (This usually means that the query is too slow.) You merely have to put the correct hint syntax just after the `select` statement to get Oracle to change its mind. For example, the following SQL statement will be optimized to obtain the first couple of rows in the fastest manner (a goal as described earlier):

```
select /*+ FIRST_ROWS */ last_name,first_name,golf_score
  from golf_scores
 where last_name='GREENE';
```

Basically, you use a construct that looks like the comment block sections in many common programming languages (`/*...*/`) with a plus sign to signify the hint (+). The key here is to know what kind of hints you can give. The most common hints are in the following list (as with everything in Oracle, there are a lot of lightly used options):

- ♦ `ALL_ROWS`: Set the optimizer goal to complete the entire query in the most rapid manner possible.
- ♦ `FIRST_ROWS`: Set the optimizer goal to get at least the first couple of rows of the query back quickly, even if it takes a little longer to retrieve all the rows.
- ♦ `FULL(table)`: Use a full table scan on the table indicated. You can use either the table name or an alias to specify the table of interest.
- ♦ `INDEX(table index)`: Use an indexed search on the specified table using the specified index. There may be cases where there are multiple indexes that could be used to find the values, and you can speed things up by choosing the right one.

- ◆ `INDEX_ASC(table index)`: Use the specified index to scan the specified table, but do it in ascending order. This will help if the values you want are likely to be at the beginning of the index.
- ◆ `INDEX_DESC(table index)`: Use the specified index to scan the specified table, but do it in descending order. This will help if the values that you want are likely to be at the end of the index.
- ◆ `AND_EQUAL (table index ...)`: Merge the results from the specified single column indexes to determine which rows of the table are needed.
- ◆ `ORDERED`: Join the tables in the order in which they appear in the `FROM` clause.
- ◆ `USE_NL (table table ...)`: Use the nested loop algorithm to join tables together, with the first table specified being the inner table on the join.
- ◆ `USE_MERGE (table table ...)`: Use the merge-join algorithm to join the tables together.

Here are a few more examples of queries that use hints:

```
select /*+ ALL_ROWS */ analysis,date_run,result
  from lab_tests
 where analysis_type = 'CHEMICAL';

select /*+ USE_NL(a) */ analysis,result,action_to_take
  from lab_tests a,test_actions b
 where a.result = b.result and
       a.analysis_type = 'CHEMICAL';

select /*+ INDEX(lab_tests lab_tests_types) */ analysis,date_run,result
  from lab_tests
 where analysis_type = 'CHEMICAL';
```

The application developers guide in your Oracle documentation set contains a pretty good discussion of the options that are available to you with hints. You can do a lot with hints to make Oracle do what you want it to do, rather than what it thinks it should be doing. The nice thing about tuning queries is that you can isolate the SQL and run multiple timing tests using a variety of hints until you find the one that works best. You do not have to worry about damaging the data as you would testing update, insert, or delete statements. You may want to be considerate about the impact that your testing might have on user response time if you are issuing some really nasty test queries.

INDEXES

Indexes are the easiest and often best technique to consider when you want to improve the performance of a query. Most of the access speed problems in new applications are due to a lack of an appropriate index. Once again, it can make a night-and-day difference in your performance. The best thing about it is that it is relatively simple to determine which indexes you need. You merely look at the SQL statements that you issue and see what you are using in the `where` clause.

You do have to be judicious regarding what you build the indexes on—indexes consume disk space that might be tight. You do not get much benefit from indexes that are almost as large as the tables themselves. Finally, indexes do have an impact on the performance of updates, insertions, and deletions from the table. With these points aside, an index can be the simplest and best way to improve query performance when you have a few key fields on which you are always searching.

GENERAL GUIDELINES

So far, you have learned that there are various execution plans and that some are better than others. You also know that you may need to try several alternatives before you find the one that works best for you. However, you do not know yet how you figure out which execution plan Oracle is using. This was intentional, because the output has little meaning until you have the basic concepts of query execution under your belt. Now you are ready to jump into the subject of analyzing how your queries are being processed.

The basic procedure is relatively simple, but there is no single good source that explains it from end to end. It begins when you run the query that you are curious about with the `trace` utility turned on. All this `trace` utility does is make an output file in the `udump` directory for the Oracle instance that contains data on how the query was attacked. (It usually is a sequentially numbered file with a `.trc` extension.) I like to extract the SQL in question and put it in a separate SQL file with the command to start the `trace` utility right before it. In that way, I can run the script from `SQL*Plus`. The command to turn on the `trace` utility is the following:

```
alter session set sql_trace = true;
```

You also need to make sure that the `TIMED_STATISTICS` parameter in the initialization file is set to `TRUE`. Once this trace file is produced (that is, you run the query), you must find out where Oracle put the trace results. You have to look through the `udump` directory to find the file that was made at the time you created and ran the query (I use the `ls -lt` command in UNIX). It is not convenient, but you have to find the

filename before you can decode the output file to produce a report. This file is not very readable. You need to use the `tkprof` utility to turn this data into something designed for human beings. Here is the format of the command that will give you this report:

```
tkprof trace_dir/filename.trc outfile.prf explain=user sys=no
```

This takes the `filename.trc` file in the `trace_dir` directory and runs it through `tkprof`. It stores the output of this process in the `outfile.prf` directory. The `user` parameter is used to connect to Oracle as a user to get some additional information. This is not the most convenient utility, but it does provide some useful outputs. The report has some general information at the top about the number of fetches performed and the time to perform the operation. It then goes into a series of execution plan results from which you can test and plan your strategies similar to the following:

```
SELECT STATEMENT
MERGE JOIN
  SORT          JOIN
    TABLE ACCESS FULL          GOLF_SCORES
  SORT          JOIN
    TABLE ACCESS FULL          GOLFERS
```

There is a UNIX shell script called “bench” included on the disk that executes a SQL script. It has the command to set tracing on and to run it through all the steps in the process. Very few people actually type all these commands at the command line.

Now let’s look at some general guidelines on the optimization process. It is really tough to come up with general guidelines for the process of query execution. However, the following steps show how I normally approach it. This should work for most cases:

1. First, make sure that your instance is properly tuned.
2. Run the query with trace mode on and run it through `tkprof` to determine how the query is being approached by Oracle.
3. If this table is looking against a few rows that are commonly used as criteria but are not in an existing index, consider building an index to solve this problem. If there is an index that it should be using, try giving it a hint to use that index.
4. If tables are joined via merge-join, try to get Oracle to use nested loops to meld the data.
5. For nested loops, try to ensure that it does the search against the smaller table first (it will be listed last in `tkprof` output).
6. When all else fails, experiment with some of the alternatives to see whether you can improve performance.

THE VALUE OF EXPERIMENTATION

By now you may be a little overwhelmed with all the options that are available to you in the query optimization world. How do you know which one will work best in your particular circumstance? There are a few people who have worked with this so much that they know right away which is the best method and whether you need to use a hint to get Oracle to follow this execution plan. I am not one of these folks and very few of the DBAs that I have met can do this off the top of their heads.

How should you figure out which is the best solution? I am trained as a physicist. Some of my fellow students sat at their desks and tried to figure out the elegant theories that explained everything. I liked to go into the laboratory and see what the reality of the situation was. Most scientists believe that theories are only as good as they match up with reality (while philosophers sit around and debate what reality is). Therefore, my favorite solution to the problem of tuning a query is to experiment with the query. I build my SQL script that turns tracing on and then run it through the process described previously to see what it is doing and how long it takes to do it. I then vary the format of the SQL (add `UNIONS` and so forth) and give it a variety of hints. After it is all over, I look at the `tkprof` outputs to see which of the various alternatives provides me with the response time that I want.

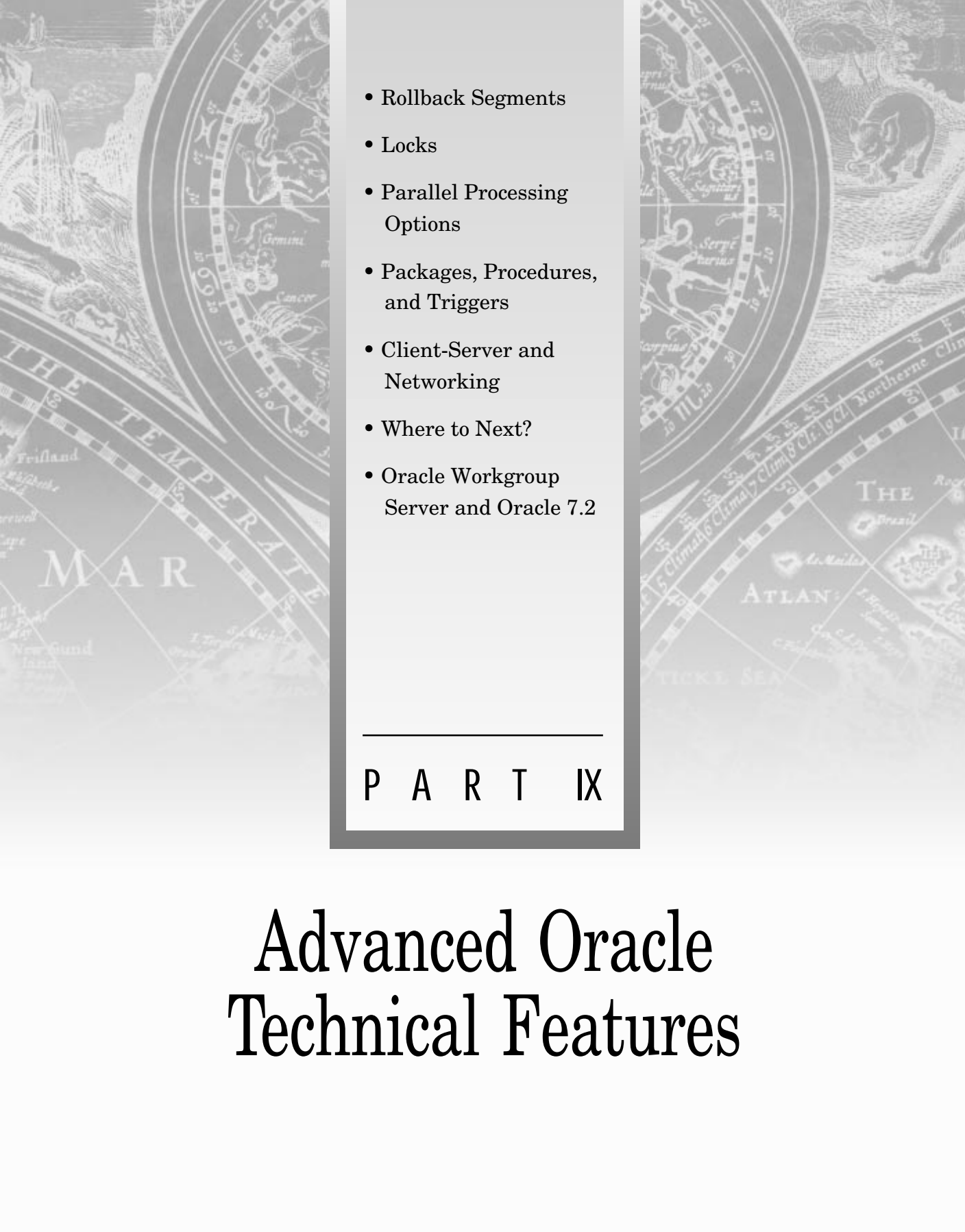
SUMMARY

This chapter covered a lot of ground on which not every DBA has to walk. It addressed the subject of adjusting queries so that they provide the best performance possible. The key to tuning a query is understanding the various factors that are under your control that affect query performance. You need to run some Oracle utilities to see how Oracle is attacking the problem and then try alternative execution plans to see which one works best. Don't feel bad if it is not obvious to you when you are looking at the code. As long as you run a good set of experiments and come up with the right answer, you've done your job.

Speaking of that, what is your job? Again, you may not have to tune queries as a DBA. Perhaps all you have to do is run your tuning reports and verify that your database is operating within the parameters recommended by Oracle. If you do help the developers when tuning their queries, you will call into play the material presented in this chapter. It's a good idea to save the results of your experiments online to be used as evidence later. It is pretty convincing when you can show the results of your tuning report, compared with Oracle performance specifications, and also show the results of all the various execution plans that Oracle can take on a given query. These can be used to convince everyone that you have done a thorough

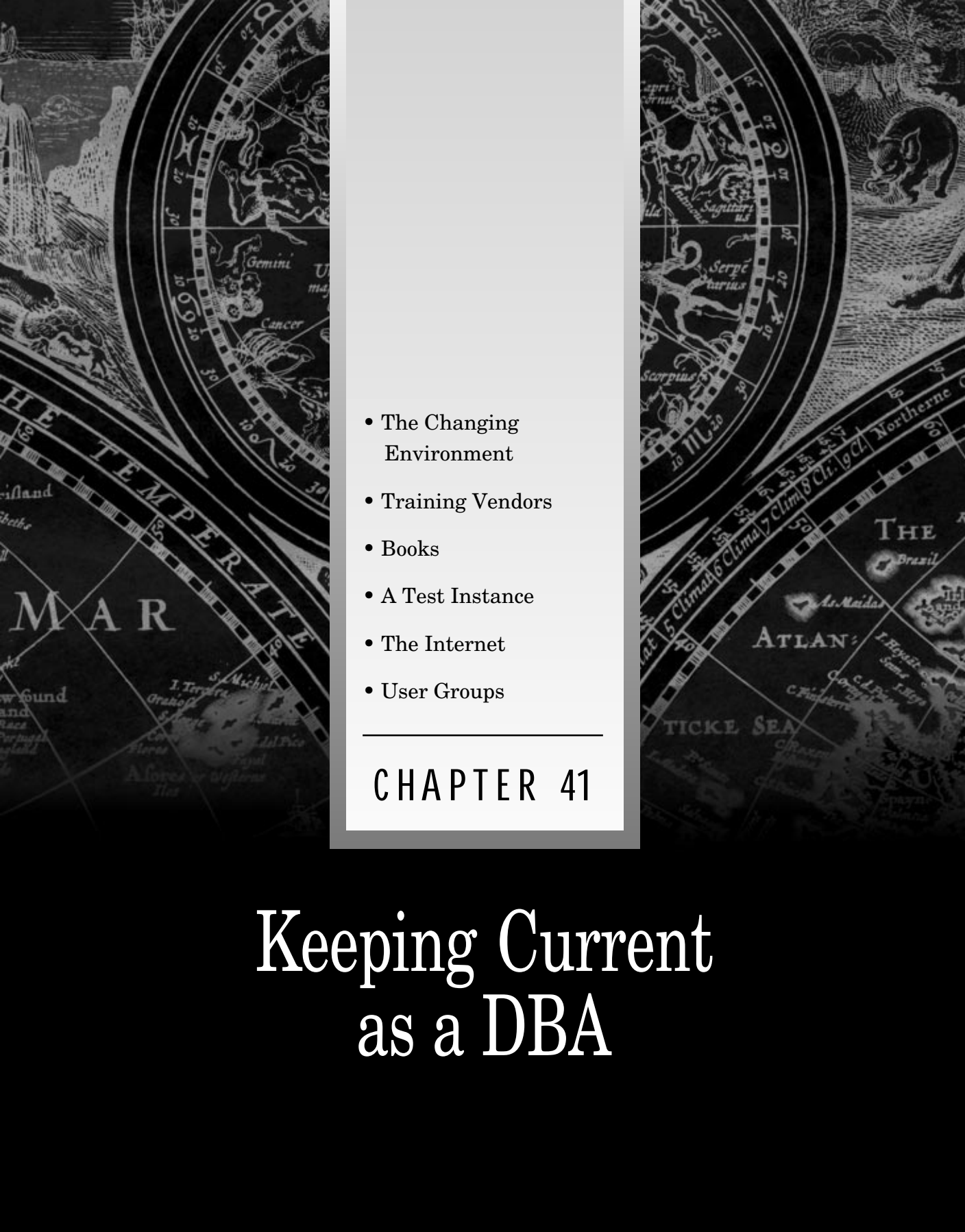
job of squeezing as much performance out of your system as possible. If you have done this thorough a job, you can feel good when you say that the system is doing as much as it can. There will be queries that take a long time merely due to the nature of what they are doing (for example, summing up a dozen columns based on a full table scan of a table containing 10 million rows, grouping the results by department).

There is one other performance improving concept that you might consider if you have a multiprocessor system—parallel queries. As described earlier in this book, Oracle 7.1 and later enables you to break a complex query into a series of subtasks that can be assigned to multiple CPUs. This way, you get a little extra help getting the job done. Not all queries are suited to this (they have to run as a single stream) and you may not want to take all the processors on your system for a single query. To determine whether you can benefit from this, you need to run some experiments (use the `PARALLEL #` hint where `#` is the number of processors you want to use in the query). Typically, you will see increased performance up to a certain point and then see it level off after that point. You should then set your hints for that number of parallel processes.

- 
- Rollback Segments
 - Locks
 - Parallel Processing Options
 - Packages, Procedures, and Triggers
 - Client-Server and Networking
 - Where to Next?
 - Oracle Workgroup Server and Oracle 7.2

P A R T IX

Advanced Oracle Technical Features

- 
- The Changing Environment
 - Training Vendors
 - Books
 - A Test Instance
 - The Internet
 - User Groups

CHAPTER 41

Keeping Current as a DBA

Although it may seem strange to have a chapter on keeping current as a DBA under the topic of supporting your users, there is not any other logical part of the book in which to put it. Placing it in this part also emphasizes that you are not keeping current as a DBA just for the fun of it or for your own personal benefit. Your ability to support your users and management depends on your keeping touch with the technology. Oracle is releasing the 7.2 release of the RDBMS within one year of releasing 7.1. They have plans to release version 8 (which will merge in some major object-oriented technologies) next year. The technology is changing very quickly.

The key point is that you cannot avoid this new technology either. If you administer your database the way you did under Oracle 6 (or perhaps still have an Oracle 6 database), you are not doing the most that you can for your users. In addition to fixing problems with new releases of software, a number of powerful features are routinely added. These features can save time and improve reliability in your database. By keeping current, you can keep everyone aware of what is going on in the real world. Perhaps management is not willing to pay for some of these improvements—but at least they can't say you didn't tell them.

This chapter begins with a discussion of the changing technology environment. I present some of the changes that I have observed over the last couple of years and then get out my crystal ball to forecast what might be coming in the future. Next some of the resources that you can use to keep current are presented:

- ♦ Training vendors
- ♦ Books (what do you expect from an author)
- ♦ A test instance
- ♦ The Internet
- ♦ Oracle user groups

By the end of this chapter, you should understand the amount of change that is taking place in the database industry. You should also be familiar with the resources that can help you keep pace with the technology. You should understand the general options provided by each of these resources and their approximate costs. From this, you should be able to create a program that you can use to keep your level of knowledge current. Of course, the exact level of knowledge that you need to maintain will vary between different DBAs (those in conservative business production data processing environments do not need to stay as current as DBAs supporting multimedia development projects).

THE CHANGING ENVIRONMENT

I really started working on Oracle with some of the releases of the Oracle 6 system that were about halfway through the life cycle of that product. They were not bad and, for the most part, worked well. We built some interesting applications (an early application to store low-resolution pictures in a database, for example) with only a little trouble setting up the networking and steps like that. However, the demands of users in the computer industry continued to grow. They wanted to build large instances with large numbers of users.

Oracle responded to these demands with Oracle 7. This product had a much better system administration scheme (roles) and some improved tools for working with the database. It also started the process of parallelizing database processes to support larger instances. This had the side effect of making a larger number of tunable parameters for the DBA to control. The goal, as it had been for some time, was to have a single database management system that supported a wide range of user needs.

During this time, the concepts of decision support systems and client-server computing became popular. Client-server architectures enabled smaller UNIX servers to take on a much larger number of users. The decision support needs (which have been called executive information systems and a number of other names by the salespeople) tended to lead to data warehouses that stored very large volumes of data. These systems also lead to very complex and demanding queries.

Oracle came up with upgraded versions of SQL*Net (2) to provide better services to client-server environments. This protocol interfaced to the multi-threaded server, which eliminated the need to have a large number of user processes idling on a server for users who were sitting and reading the output that they just created. It also provided additional communications reliability features to maintain good connections and disconnect bad connections. Combined with the growth of other middleware and enabling technologies (for example, the Open Database Connection or ODBC standard), client-server has become a very popular development methodology for new applications.

To support the very large queries that data warehouses can produce, Oracle introduced Oracle 7.1, which enabled you to divide a query into multiple tasks. These tasks could be assigned to multiple CPUs in multi-CPU computers. This can dramatically improve performance of some queries. It is also in line with a major theme in the computer industry toward parallel processing. It is becoming much more difficult to increase the performance of a single computer chip. However, if you get a large number of processors working together, you can achieve very high levels of performance. Oracle has invested heavily in parallel technologies.

Now for the fun stuff. If I look into my crystal ball (based on reading computer magazines and similar stuff), I see the following vision for the Oracle environment. I see larger parallel processors offered by almost every major server vendor. I see Oracle moving toward object-oriented technologies that enable it to store both the methods and the data together (slowly at first, but then eventually becoming very object-oriented). I see more client-server applications and improved networks to handle these demands (everyone likes a nice windows interface). I see Oracle DBAs continuing to make a decent salary (okay, perhaps I'm biased on that one).

Seriously, I have seen a large change in my job as a DBA. My first system held a few users and had little in the way of security requirements. As time went on, I saw larger systems that demanded very fine control over the security of data. Oracle has changed its tools to enable me to better meet these challenges (especially roles in Oracle 7, which make administration almost bearable). As the instances that I have been working on have moved more toward business production environments, I worked to develop administration schemes, reports, and formal procedures to get the job done in a more proactive manner. It was okay to be down for a few hours on that engineering system, but not on a key user transaction processing system.

I am also firmly convinced that the changes will continue well into the future. Not just in the technology, but in how I do my job. There are a number of third-party tools that are coming out to help DBAs get their jobs done. These, along with upgraded administration tools from Oracle, need to be continuously evaluated. You will probably also have to become familiar with the workgroups and personal versions of the Oracle RDBMS (Windows NT, OS/2, and MS Windows versions). You will probably even have to deal with distributed database applications (where some of the data is on the local machine and the rest of it is on one or more servers). Your job is to figure out how you will keep up with these changes.

TRAINING VENDORS

One of the great traditions in the computer industry is the training program offered by a vendor of a computer product. In the early days, when no one other than the manufacturer understood the products, computer types would jump into planes and fly off to California or Boston for a week of training. Once this training was completed, they were certified as local gurus on the product and ready to answer all questions. Many argued that a number of these training programs thrived because of manuals that were incomprehensible to mere mortals. Whatever the case, it was an important career step and considered normal business practice in most computer shops.

As the computer industry grew, the vendor training programs began to spread out across the nation. They also added a variety of courses to meet various user needs. They still tended to be somewhat expensive (several thousand dollars per week), but became more reasonable as volumes increased and travel costs were minimized for those who could travel shorter distances. There were a number of oldtimers who missed the opportunity to fly out to Silicon Valley or enjoy seafood in Boston, but costs were being controlled.

Another trend that grew up in this more cost-conscious era was third-party training courses. As computer products were used in the real world, a number of nonvendor people became quite skilled in the use of various products. They and some entrepreneurial ex-vendor staff members saw the market opportunity and formed small companies to offer training on various computer products. Some of these programs were far better than what was offered by the vendors, and they cost less. It also spurred vendors into upgrading their training programs, offering them in more locations and lowering their costs.

So what are you faced with today? If your Oracle sales representative is worth his keep, he has probably ensured that you are at least aware of the training programs that Oracle has available. The factors that you have to consider with these programs are their costs (they are usually more expensive than those offered by smaller, local firms), their quality, and whether the courses you want are offered in convenient locations. They obviously cannot offer every course at every location every week. If you live in an area such as Washington, D.C., you have access to a wide range of courses offered frequently. If you live in Pittsburgh, you will find the common courses offered frequently, but you may have to travel if you want a particular DBA course before a certain date.

You probably also have a mailbox filled with catalogs from local and national training vendors who offer competing courses. They may have more courses available in the local area and may be more willing to offer customized or on-site training at reasonable costs. You should check the quality of these courses before you commit to anything (sit in on one or talk to other people who have attended these classes). You should also be concerned about the possibility that a particular class may be canceled if attendance is too low. You do not want to find out two weeks before you install Oracle that all your training has been canceled due to low enrollment.

As if the range of training courses was not wide enough, a number of alternative technologies are available to train you on Oracle and other technical subjects. I worked with some early computer-based training systems about 10 years ago (video disks linked to PCs). They were enormously inflexible and expensive. However,

since then the technologies have made computer-based training a reality. You can now have audio, video, and data on the screens of computers that cost merely \$1,000. The video industry also puts out tapes that contain material similar to those of the in-person classes.

There are a number of pros and cons to these automated learning technologies. First, you can take computer-based training and videotape training classes whenever you can fit them in your schedule. You can even take many of them at home if you have a VCR and PC. This can also be the negative side to the issue, especially if you have little time at home already. Additionally, there are a lot of people who need to be able to ask questions and actually work with the product to learn. However, these products typically cost less than live presentations, especially if you have a number of people to put through the training.

In summary, there are a number of training vendor options. When you get to a specialized area such as database administration, you will probably tend to go more for live presentations from one of the larger training vendors. However, you may want to keep the other options in mind if you are limited by budget or lack of ability to get away from the office for a week of training. The option of doing some of the training at a vendor site and following up with specialized topics (SQL*Net) through videotape programs might also work.

BOOKS

It costs a lot less to buy a couple of books at your local bookstore than to attend one of the training programs. Another advantage is that you can look through the curriculum and presentation before committing to it (training catalogs rarely show presentation quality). However, you do not have anyone to ask questions of when you run into problems. I tend to buy computer books when I need to come up to speed on the topic. My collection exceeds the book shelf capacity and has actually sprawled onto the floor. If I have questions, I usually can find some expert in the field who is willing to talk to me, and there's always the Internet with its wealth of experts.

This book would make quite a thud if you dropped it on the floor and would probably be considered a deadly weapon if you hit someone with it. However, in spite of its size, it cannot cover every topic that a DBA might need. Specialists in large databases might want to consider one of the books on Oracle tuning. If you lack network expertise, a book on client-server computer or computer network technologies might help you. There even are books on dealing with difficult customers if you find you need to branch out from the purely technical computer disciplines. Books are a good resource for those who do not have the budget or time to attend a large number of live training courses.

A TEST INSTANCE

One of the best things that you can do is create a test instance for yourself and your developers. There are just so many DBA tasks that you should not perform for the first time in a production instance. The classic example is Oracle software upgrades. You do not want to have your production instance down for several days while you wait for the tape that will enable you to complete the upgrade. Of course, you will have backup tapes, but it may take time to perform the restoration that your users are not willing to give you. You may also find that you complete the upgrade and go along for several days before you find a really nasty bug. What do you do—lose several days worth of data or try to live with the bug?

Although upgrades are the most obvious thing that a test instance can be helpful for, they are not the only ones. You have to remember that when you issue commands as a DBA, you have the Keys to the Kingdom. You can delete things, overwrite things, and generally render your instance unusable. People are usually a lot more tolerant of a down test instance than a down critical production instance. You also have to remember that developers can issue some really nasty queries and fill up tablespaces without even thinking about it (and these problems usually result when they are not thinking). Giving them a place to work out all the bugs with their applications is good for them as well as the production users.

If you cannot afford the disk space or processing capacity of keeping a full-time test instance, consider creating one just for quick testing. If you have enough disk space but lack processing capacity, bring this special test instance up only when you need it. You can also create one on a small set of disks just before you need it and then delete it before restoring normal processing. This can be an especially useful technique when testing whether the Oracle upgrade utilities are working properly and you do not want to impact your developers. Whatever your needs, have a test instance available (no one will ever see the mistakes made on a test instance).

THE INTERNET

The Internet is a topic that can fill several volumes. It is also one of the hardest topics to write about because it changes at such an enormous pace. Recall the discussion about how rapidly the new releases of the Oracle software were coming. Imagine the pace of information change on the World Wide Web, which doubles the number of sites every 53 days (at least that was the last statistic I heard before writing this). You can usually keep up with the basic technologies used to exchange information, but only the heavy duty web “surfers” can even begin to keep up with where all the information is located. Actually, there are a number of projects that are designed to automate the cataloging of information sources on the Net. I have a few favorite

locations that I keep stored in my list of sites; then I access one of these tools (for example, the Yahoo server at <http://www.yahoo.com>) to search for material on other topics.

One of the most basic resources on the Internet is electronic mail. It is much like the electronic mail that you have in your office, but you can connect to a huge number of people around the world. I have exchanged e-mail on Oracle topics with people from the Soviet Union to Brazil. Oracle people are also available on the net. I have actually had Oracle tech support people encapsulate a patch (a small one) to the Oracle software and send it to me as an attachment to an e-mail. I had the patch in less than two hours. E-mail is also a good way to extend the circle of colleagues who can help when you run into problems.

A good resource to use to keep up with what is going on are the Internet newsgroups. These range from a number of personal topics (for example, philosophical debates of space program policies) to discussions on most common computer technologies. Newsgroups are areas that hold a number of messages that are focused on a particular topic (postings and their responses). You merely obtain a newsgroup viewer that is attached to the network and specify the name of the newsgroup to which you want to subscribe. Subscriptions are free, and there are even tools that will help you find newsgroups that you might be interested in. Oracle is represented in the `comp.database.oracle` newsgroup. This group usually has over 100 postings per day. You usually wind up scanning a screen with headlines of the postings and then choose the ones that are most interesting to you. You can also find operating system-specific newsgroups if you are interested in keeping up with your operating system.

Another interesting resource is the ftp (file transfer protocol) sites that are on the net. If you have multiple UNIX computers in your organization, you have probably used ftp to transfer files between computers. Believe it or not, there are sites on the net that allocate disk space to store useful files and then enable other people to copy these files. You have to figure out the addresses of these sites and how to log in (they usually provide *guest accounts* with default passwords). You can use one of the Internet directory tools to find ftp sites or talk to your friends. There are a number of sites with Oracle utilities that you might be interested in. Here are two:

- ♦ [ftp.informatik.uni-oldenburg.de/pub/oracle](ftp://informatik.uni-oldenburg.de/pub/oracle)
- ♦ [ftp.ml.csiro.au/software/jstander/oracle_utilities](ftp://ml.csiro.au/software/jstander/oracle_utilities)

Now on to the hottest topic on the Internet today—the *World Wide Web* (WWW). It is based on some interesting research and development activities related to the use of compound text and graphics conducted at a number of sites over the last decade. Basically, you have sites that provide a convenient, often colorful, access tool to enable you to get information from their systems. It starts with a main menu

referred to as the *home page*. On it you find text, pictures, and icons. There are usually some *hot areas* where you can click your mouse to access additional information. Oracle even provides (free, if you can believe it) a set of tools that can link a Web page to an Oracle database (for which, of course, Oracle charges). This technology is exploding, but you can find a huge amount of pictures, text files, and even sound clips that you can view and download to your computer. All with a really easy GUI interface. Here are two interesting sites related to Oracle on the Web:

- ◆ <http://www.oracle.com/> (Oracle corporation itself)
- ◆ <http://www.ioug.org/> (International Oracle Users Group)

There are practical concerns that you have to consider about the Internet. First, you have to pay for a connection to it. Perhaps you are already on the network (most universities and many government agencies are already connected). A growing number of companies are connected to the network also. If you are not connected, there are probably local providers who would just love to connect you for a fee. You can choose either to purchase an Internet tool set or download some freebies from the Internet. Most companies are also concerned about security when connecting their networks to the Internet. You have two basic options here. You can either buy a security system known as a *firewall* to protect your network from hackers or you can use a dialup service that connects only a single PC to the network—and only when you want to make the connection. The PC dialup connections can be obtained from most of the large network providers (CompuServe, America Online, Prodigy, and so forth), and they usually cost much less than a direct connection (my bills run between \$10 and \$50 most months).

If you are really interested in the Internet, there are entire books on it, which are probably located very close to where you picked up this book. Be sure to select a book that is current. It is fascinating watching this community evolve before our eyes. Remember that there are a number of tools on the Internet that can help connect you with other DBAs, source code for tools, and other resources (including Oracle itself). It is helpful not only for Oracle DBAs, but for most computer professionals in general. (Your operating system administrator, software developers, and hardware techies can probably get information from their favorite vendors on the net.)

USER GROUPS

A nonelectronic and usually pretty inexpensive resource that you can tap are the Oracle user groups. They range in scope from the local user groups who meet at various member company sites to the International Oracle User's Group. If nothing else, you can usually find people who are dealing with the same problems that you are when you participate in these groups. You may also find some really knowledge-

able folks who may be willing to help you when you get into a bind. Another key benefit is that these groups often invite vendor representatives who demonstrate and discuss their products. You can get a quick overview of the product at one of these meetings and follow up with only those vendors who interest you (no months of phone calls from someone wanting to sell you something that you do not need).

One of the best things about these meetings is the discussions about problems people are having. You can usually get some really good ideas from other members of the group. It can be a challenge to see whether you can think up solutions to problems before they happen to you. You will also get a feel as to whether a particular upgrade might not be ready or might be a good idea for you. If you have a good group, these sessions provide valuable information and may even stimulate some brain activity.

SUMMARY

This was a chapter devoted to you. It is all right to think about yourself every now and again. You need to have training to keep current as a DBA. The Internet and users groups can serve as resources when you need information. Finally, computer books can enhance your level of knowledge. Taken together, you can usually find a means of keeping current that fits your schedule and budget. Remember that keeping current not only helps keep you employable (an important concept these days) but also makes you better able to serve your users.

This completes this part of the book on Supporting Users and Developers. There were three major topics in this section. First, you learned about object design that enables you to support users and developers who are struggling with how to store data in a manner that works with Oracle rather than against it. Next, you learned how to make queries run a little faster. Finally, in this chapter, you explored a few tools that you can call upon for support and to increase your level of knowledge.

The final part of the book is called Advanced Oracle Technical Features. Material that goes beyond the minimum theory needed to get your database going and productive is presented in this part. Some of these topics, such as locks and packages, may not be problems for many DBAs. However, it is presented for those who need it. Think of these chapters as going a little beyond mere survival to living comfortably.

-

- Special Storage Considerations
- Common and Confusing Error Messages
- Setting Up Rollback Segments

CHAPTER 42

[illegible]

Welcome to the final part of this book, “Advanced Oracle Technical Features.” If you are a “non-advanced” DBA, don’t worry. The word “Advanced” merely indicates that you may not need the material in these chapters for basic survival. Review this material to gain a deeper understanding of subjects that may help you improve your database.

There are so many little topics that you could dig into in this part. The following are those that are most likely to be of use to the average DBA:

- ♦ Rollback segments
- ♦ Locks
- ♦ Parallel processing options
- ♦ Packages, procedures, and triggers
- ♦ Client-server and networking
- ♦ Where to go next
- ♦ Oracle Workgroup Server and Oracle 7.2

This chapter begins the discussion with rollback segments. You have already explored most of the basic concepts of rollback segments. Recall that they store transactions that are in progress. If you have moderate-sized transactions where the commits occur at regular intervals, you will probably never have to fuss with rollback segments. However, if you have large batch updates or your developers do not commit routinely in their applications, you need to know a little about these specialized database objects.

The chapter starts with a brief review of how rollback segments work, then presents some of the storage issues associated with them (which are the major reasons why a DBA has to worry about them). Next, you explore the error messages related to rollbacks, which are often confusing. Finally, the chapter wraps up with some general guidelines to follow.

INTRODUCTION

You already had your basic overview of rollback segments in the early chapters on the technical foundations of Oracle. Even though you may remember every detail of that discussion, here’s a quick review for those who flipped right to the part of the book that had the word Advanced in its title.

Rollback segments hold transactions that are in process. They are used to back out transactions that may have been applied to the database due to a lack of room in the database buffer cache in memory but are not committed. As such, they are extremely important to Oracle’s plans for data integrity. They are the feature that ensures that you have to issue a commit statement before your changes become permanent.

Whenever you are dealing with something that affects data integrity within the Oracle system, you usually find software that insists that everything be right or it cancels everything. Such is the case with rollbacks. They are also extremely sensitive to storage space problems in instances that process large transactions (or fail to commit frequently). For the most part, these objects remain hidden from view. You work directly with the tables and indexes while the rollback segments are working in the background. Unless they receive an error message that mentions rollbacks, most users are unaware of their existence. Therefore, they are likely to come to you lost and looking for answers.

Most modern Oracle installations (7.0 and later) come with a separate tablespace for rollback segments. If you use the Oracle utilities to create a database, you automatically get from two to four rollback segments created for you. They are sized to accommodate most of your needs. If your transaction volumes and the size of your transactions is moderate, you will probably never deal with rollback segments.

There are two types of rollback segments—public and private. Private rollback segments can be used by only one Oracle instance. Public rollback segments can be used by multiple Oracle instances. Public rollback segments are the primary type of rollback segment that you have to worry about as the DBA. It's a good idea to create these by default because it is annoying to make modifications to the rollback segment configuration.

There is, however, one nasty little trick that you have to consider. You always create one rollback segment owned by the SYS user when you create a database. The problem is that this rollback segment is not shared. You have to drop and re-create rollback segments to change their parameters. However, if you want to re-create all your rollback segments, you have to ensure that you keep at least one rollback segment on line at all times so that you can issue the transaction needed to create the other rollback segments. If you do not have access to a rollback segment, you have to do your work as SYS (and I always avoid using that account whenever possible because it owns all the internal Oracle tables and you can make *big* mistakes when using it).

There is not much else to say about them. Unless you specify a specific rollback segment in your SQL commands, you will be assigned to a rollback segment on a rotating basis (much like the online redo logs). They have storage parameters that, with one exception, are similar to those used for tables and indexes. That storage exception is covered in the next section. Otherwise, you can just think of rollback segments as temporary tables holding transactions in progress.

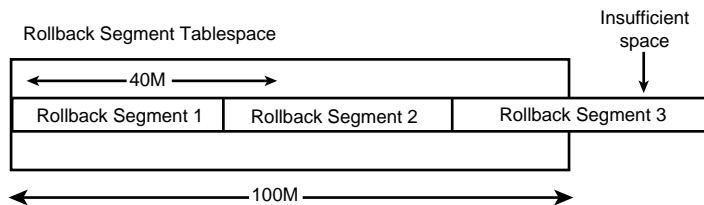
SPECIAL STORAGE CONSIDERATIONS

Storage considerations usually catch your attention when you try to perform a large data transfer into the Oracle database. A classic example might be importing a table from an export file or tape. The problem comes when you need to allocate a large amount of space in the rollback segment tablespace to accommodate this. There are two nasty quirks in the default rollback storage parameters that make this a mess.

The first problem is that rollback segments do not, by default, clean up after themselves. If you allocate a large amount of space in the temporary tablespace when performing a sort, that space is released for other users when you are done. Rollback segments consume space and do not give it back unless you tell them to specifically. This is compounded by the second problem: Oracle allocates a specific number of rollback segments and uses them in a cyclic order.

Suppose you are performing a number of 40M table imports in an instance where you have three rollback segments in a 100M tablespace. You say, “No problem, I have 100M and will be using only 40 at a time.” Wrong! This is how it works: Your first import consumes 40M in rollback segment one. Your second import consumes 40M in rollback segment two. Neither of these rollback segments gives up their space. Therefore, your last import of 40M bombs out because the third rollback segment can take up only 20M. Worse yet, if you have a series of 30M transactions to run, you will fail one out of every three tries when you are sequentially assigned to that old rollback segment three (see Figure 42.1).

Figure 42.1.
Allocation of space in
rollback segments.



There are two solutions. You can create rollback segments of different initial sizes and specifically assign certain transactions to these segments. This can be cumbersome and you may have to change all your application software if you have to re-allocate rollback segment sizes as the database grows and changes. The easier solution, and the one that I recommend, is to use an optional parameter that specifies the size to which rollback segments should shrink when they are not in use and other rollback segments need space. That storage parameter, unique to rollback segments in Oracle version 7.0 and later, is the `OPTIMAL` parameter. There is one trick to this parameter (you know that these tricks are what keeps DBAs in business). You need to ensure that the optimal setting is at least as large as the initial extent plus the first next extent. Let's look at a sample `create public rollback segment` command:

```
create public rollback segment r05
  tablespace rbs
  storage (initial 1M
    next 1M
    pctincrease 0
    optimal 2M
    minextents 2
    maxextents 100);
```

Most of this should look pretty normal except for the storage parameters. You need to have at least two extents in a rollback segment. (I did not make up these rules, Oracle did—I just live with them.) Therefore, you need to set your `optimal` parameter in this case at 2M. The only other parameter that you need to set is the `tablespace`.

Before you leave the subject of rollback storage, here's a tip on dropping and using rollback segments. If you try to drop one of the rollback segments that is in use in your database, you will fail. That is because rollback segments, much like tablespaces, need to be taken offline before you can do any work with them. When you first create a rollback segment, it is offline unless you specify `online` in your `create` command. Therefore, a session to re-create a rollback segment to make it use larger segments might look something like this:

```
alter rollback segment r02 offline;

drop rollback segment r02;

create rollback segment r02
  tablespace rbs
  storage (initial 1M
    next 1M
    pctincrease 0
    optimal 2M
    maxextents 100);

alter rollback segment r02 online;
```

It may seem like a bit of a pain when you are first dealing with these beasts. The hardest part for me was understanding that they do not clean up after themselves. It took me a while to figure this out because it was so unlike temporary tablespace usage. Once you figure this out, all you have to remember is the `optimal` parameter and that you have to take the segments offline to work on them. Okay—maybe some practice working with rollbacks will help, too.

COMMON AND CONFUSING ERROR MESSAGES

Most people find Oracle error messages confusing. Because most people are not aware that there is such a thing as a rollback segment, they find these error messages doubly confusing. The messages sometimes point to a symptom of the problem rather than to the problem itself. For example, `Failed to allocate an extent` may not mean that there is a lack of space; it may mean that the tablespace is just

too fragmented. Most people find that there are two common rollback segment-related error messages.

The first of these messages goes something like `Failed to extent rollback segment (ID = 001)`. This is usually an indication that you have run out of space in the tablespace containing that rollback segment. It could mean one of two things. First, you might have a really huge transaction that is taking up all the space in that tablespace. All you can do in this case is cut down the size of the transaction (commit more frequently) or increase the size of the rollback segment's tablespace.

Another possibility raised by this error message is that you have not set up your initial parameters for one or more of the other rollback segments in the tablespace. These other rollback segments have probably grown to fill up the entire tablespace and left no room for their friends. The solution to this is to set the initial parameter for all your rollback segments. To determine what the cause of your problem is, look at the output of the `v$rollstat` view. You are looking for the `hwmsize` (high water mark size, which is the most space that this rollback segment consumed). See whether this value is approaching the size of the tablespace (in which case you are out of room). Check to see how many extents this high water mark corresponds to (you may have exceeded your `maxextents` storage parameter). Look also at the `optsize` parameters to make sure that they are set to reasonable values. The following is a sample of the output for this command:

```
SQL> select * from v$rollstat;
```

USN	EXTENTS	RSSIZE	WRITES	XACTS	GETS	WAITS	OPTSIZE	HWMsiz
WRAPS	EXTENDS	AVESHRINK	AVEACTIVE	STATUS				
0	4	202752	1012	0	16	0		202752
0	0	0	0	ONLINE				
1	53	540672	819	0	8	0		540672
0	0	0	0	ONLINE				
2	2	202752	3217	0	13	0		202752
0	0	0	0	ONLINE				
3	2	202752	1221	0	8	0		202752
0	0	0	0	ONLINE				

The third possibility raised by that error message is that your rollback segment is very fragmented. Although there is plenty of space to store your current transaction, you cannot get enough empty segments that are large enough to accommodate your rollback segment extent size. This often happens when you have mixed sizes for your rollback segments or use a `PCTINCREASE` of other than 0. The solution to this problem is to de-fragment your rollback segment tablespace by dropping and re-creating all the rollback segments within it.

Another set of problems that you can run into with rollback segments is indicated by an error message that goes something like `Failed to acquire rollback segment...`. There are a number of messages that relate to this, but they usually result from having too few rollback segments available. Oracle is trained not to wait too long to process a transaction. Therefore, if it cannot get a rollback segment in a reasonable amount of time, it aborts the transaction and gives the error message. Your solution is to create additional rollback segments in your instance.

A final problem that you may encounter with rollback segments is the `Snapshot too old` error message. There are four common causes of this problem:

- ◆ If you have a lot of transactions, you may have rollback segments that are too small for the number and size of the transactions.
- ◆ You could have a corrupted rollback segment.
- ◆ You could have too few checkpoints in trusted Oracle.
- ◆ You may not be closing cursors on a long-running query. This problem can be fixed by increasing the size of the extents for the rollback segment. Once again, it means dropping and re-creating the rollback segments to change their storage parameters.

SETTING UP ROLLBACK SEGMENTS

Quick and dirty—how do you set up rollback segments? I usually accept the number of rollback segments created by default. I increase this number only if I run across an error message or anticipate a large number of transactions occurring at the exact same time. Then I set the `maxextents` parameters to match those of my other database objects, which is a little lower than the total `maxextents` allowed by my version of Oracle. For example, if my version of Oracle allows 256 extents, I may set `maxextents` at 225 or even 200 so that I can raise it quickly if I run into a problem.

If you use this plan, you next have to figure the size of the extents that you need. Figure out the size of the largest transaction that you will be doing. This usually corresponds to the largest import that you might perform to de-fragment a table. This may be impossible in data warehouses (try to de-fragment a 17G table). However, this is a good starting point for most instances. Try to size the extent so that you can perform this import without exceeding the `maxextents` parameter and with a little safety margin. For example, if the largest table that you would de-fragment is 400M and you have 225 as your `maxextents` parameter, you might choose 2M as your initial and next extent size. This will accommodate an import of a table that is up to 450M (always leave yourself a little safety margin).

The final recommendation on setting up rollback segments is always to use the `optimal` parameter. Set it up so that you consume exactly the initial plus one next extent. This keeps things simple, and having Oracle clean up after itself is much easier than controlling the applications. I have never had any trouble with the `optimal` parameter's functioning. I have had a number of problems when the `optimal` parameter was not set.

SUMMARY

This chapter on rollback segments is the first in a series of chapters devoted to some of the more advanced concepts related to an Oracle database. These database objects usually work well without any intervention or even human awareness of their existence. However, if you have large transactions (such as imports of large tables), you may run into trouble because rollback segments do not clean up after themselves. You can correct this problem rather quickly by using the `optimal` parameter. This chapter also provided advice on setting up the storage parameters for the rollback segments in your databases. Here's hoping that you never have to deal with your rollback segments.



- Overview
- Types of Locks Applied
- Freeing Locks
- Determining When Application is Waiting for a Lock to Release

CHAPTER 43

Locks



This chapter, the second in the series of quick discussions about some of the more advanced technical concepts within Oracle, centers around the locks used by Oracle for transaction integrity. This chapter presents a basic overview of locks within Oracle and what types of locks are applied, either by Oracle itself or overtly by the programmer as part of the software. You learn how to free these locks and how to determine quickly what locks are being applied so that you can tell when you have a problem.

OVERVIEW

Single-user databases have it easy. There is only one user who can work on the system. That user is instantly committed and the state of the database is fixed based on that one user's actions. With multi-user databases, you have to be careful to ensure that they provide a consistent and concurrent picture of the data in the database. Consistent means that a user always gets the latest picture of the database based on all the transactions that have been committed (not just entered) to the database at the time the query is issued. Concurrent means that the activities of multiple users are coordinated to ensure that the instance provides good functionality and provides correct answers and processing.

There are a number of things that can go on in the database that can cause problems with consistency and concurrency. The following is just a partial list of these problems:

- ◆ You do not want to allow someone to perform an operation, such as dropping a table in which someone is currently inserting data.
- ◆ You do not want to read any data that has not been committed as of the time of the query.
- ◆ You do not want to have one update overwrite the data of another update until the first update is committed.
- ◆ You do not want to miss data from a transaction that has been committed but is still in the process of being written to disk by the database writer process.

To do this, Oracle keeps track of rows in various database objects that are in a period of change. Some things do not require locks. Oracle internally decides what needs locks and applies them without your having to worry about it. The first concepts that you have to get used to are the ideas of *exclusive* and *shared lock*. If someone has an exclusive lock on a resource (table, index, and so forth), no other user can access that resource. Shared locks, on the other hand, enable resources to be shared if the competing process does not require an operation that, in turn, requires an exclusive

lock on that resource. Oracle tries to use shared locks whenever possible to permit maximum availability of database resources.

Oracle also tries to use the minimum amount of locking possible to promote access while ensuring data quality. Oracle has the capability of either locking individual rows or the entire table, depending on what is needed. Oracle uses the following rules to determine how much locking is needed:

- ◆ You can read the current copy of a given row even if there is a transaction pending that will update that row.
- ◆ You can write to a row without waiting for someone to finish reading that row.
- ◆ You can write to a row unless you are trying to update the same row that someone else is already in the process of updating.

There is one final point to consider: Many of the SQL statements that the DBA issues (such as `create table` or `alter rollback segment`) are actually transactions to the database. They interact with all those data dictionary tables that are owned by the SYS user and rarely seen by outsiders. However, because many of these transactions (for example, `alter tablespace add datafile ...`) can run for quite some time, you may lock out your ability to perform other DBA activities on that object while you are waiting for the first transaction to complete. Note that it does not block access to tables within the tablespace; it blocks access only to the rows in the data dictionary that relate to the object.

TYPES OF LOCKS APPLIED

The first type of lock that you need to consider is the one that is applied to ensure that read operations are consistent. There are two basic options—read-only locking and exclusive table and row locks. Read-only locking allows the users to read the data as it existed before the transaction in progress was begun. This ensures that everyone obtains the same results for their queries until the transactions in progress are committed (at which time they see the new values). The exclusive row and table locks prevent anyone from touching this data until the transaction in progress is either committed or rolled back.

There are a variety of table locks provided by Oracle. They are designed to ensure that the minimum amount of restrictions are applied so that the data is not compromised. The following table locks are provided by Oracle:

- ◆ `RS` allows other transactions to perform modifications to rows in the same table. The only transactions that are prohibited are those that require exclusive locks for the entire table.

- ◆ `RX` exclusively locks one or more rows in the table, thereby preventing other transactions from affecting the specified rows, but allowing transactions to work on other rows within the table.
- ◆ `s` prevents exclusive locks from being applied to the table.
- ◆ `SRX` (share row exclusive) exclusively locks rows in the table.
- ◆ `x` locks the table exclusively and prohibits transactions to the table. It does permit queries to the table.

The following table shows how some common statements apply locks to the tables. Note that it is not very restrictive in allowing other users to access other rows in the table for transactions.

<i>SQL Statement</i>	<i>Lock Applied</i>
<code>SELECT...FROM table ...</code>	None
<code>INSERT INTO table ...</code>	<code>RX</code>
<code>UPDATE table ...</code>	<code>RX</code>
<code>DELETE FROM table ...</code>	<code>RX</code>
<code>SELECT ... FROM table</code> <code>FOR UPDATE OF ...</code>	<code>RS</code>

Another way to look at locks is to consider whether they are applied manually or automatically. Although the vast majority of locking is performed automatically by Oracle, you do have the option of applying the locks previously described. The following statements can be used to apply locks to the `golf_scores` table:

```
lock table golf_scores in row share mode;

lock table golf_scores in row exclusive mode;

lock table golf_scores in share mode;

lock table golf_scores in share row exclusive mode;

lock table golf_scores in exclusive mode;
```

There is one “gotcha” that I have run across in tables that use unique indexes (primary keys and so forth). The question comes up as to how you can guarantee the uniqueness of the key if two transactions (updates, for example) are in progress to the table at the same time. If they do not know about one another, there is no way that they can guarantee that they will not insert the same values for that primary key. In this case, I have observed Oracle locking the index until the first transaction completes. This makes sense, but you need to keep an eye out for this if you run a lot of large transactions in parallel on tables that use unique indexes.

FREEING LOCKS

There are two basic ways to free locks that you have applied to a database object—commit the transaction or roll it back. This can be a problem if you have a large number of transactions waiting for the first transaction to complete. When a lock on the table or rows is freed, the next transaction or query can access it and apply its own locks. As you can see, you can build a large backlog of transactions if they need to perform exclusive types of locks on a frequently used table.

There is another condition that is even worse than having a long queue—deadlock. Suppose, for example, you have two SQL scripts that are each three lines long. They work on the same rows of the same table, but in reverse order:

Script 1

```
update golf_scores
  set score = 72
  where golfer = 'JGREENE';
```

```
update golf_scores
  set score = 102
  where golfer = 'BSMITH';
```

```
commit;
```

Script 2

```
update golf_scores
  set score = 105
  where golfer = 'BSMITH';
```

```
update golf_scores
  set score = 69
  where golfer = 'JGREENE';
```

```
commit;
```

If these two scripts are executed at the same time, you run into problems. Script 1 locks the row for JGREENE and Script 2 locks the row for BSMITH. Neither of the two scripts can execute its second statement because that row is locked by the other script. This also means that they cannot get to the `commit` statement that is needed to free the deadlock. Fortunately, Oracle is smart enough to detect this situation in many cases and will stop one of the offending processes with an error message. It beats having the two processes sitting around waiting until the end of time while the users go to the DBA to complain about response time.

DETERMINING WHEN AN APPLICATION IS WAITING FOR A LOCK TO RELEASE

What happens when you are not quite in a deadlock situation, but a number of processes are competing for locks on a given resource? The applications involved slow down and users go to the DBA complaining about response time. If you suspect a lot of transactions are trying to apply locks to the same resource (especially locks applied to unique indexes), you may want to use the `monitor locks` option of `SQL*DBA`. You look for a lot of resources asking for exclusive locks to the same resource number. If you see this happening, check to see what applications are running and what can be done to free those locks (more frequent commits, balancing when these jobs are run, and so forth).

You can also issue queries against the database from tools such as SQL*Plus. The following is an example of a query that shows that I had a transaction exclusive lock on a table (I updated every row of the table, but had not committed it yet). Note that there are a number of locks applied by Oracle for its own internal purposes (media recovery, redo threads, and so forth):

```
SQL> select * from sys.dba_lock;
```

SESSION_ID	LOCK_TYPE	MODE_HELD	MODE_REQUESTED	LOCK_ID1	LOCK_ID2
2	Media Recovery	Share	None	5	0
2	Media Recovery	Share	None	4	0
2	Media Recovery	Share	None	2	0
2	Media Recovery	Share	None	3	0
2	Media Recovery	Share	None	1	0
3	Redo Thread	Exclusive	None		0
6	DML	Row-X (SX)	None	1046	0
6	Transaction	Exclusive	None	65538	173

8 rows selected.

SUMMARY

There is more to learn about locks. The server concepts manual devotes an entire chapter to the subject. This chapter provides you with a basic overview of the subject so that you can detect when locks are causing you problems. You explored the types of locks that are available and what they mean to other transactions trying to access that table. You also learned to use `monitor locks` within SQL*DBA to find out whether there are a number of transactions competing for the same resource. If you can do this, you probably know enough about locks to be a very successful DBA.

- 
- Types of Parallel Processing
 - When to Use Parallel Processing
 - Distributed Databases Versus Parallel Servers
 - Multi-Threaded Servers
 - Asynchronous Database Writers
 - Parallel Query
 - Parallel Recovery
 - Summary

CHAPTER 44

Parallel Processing Options

How can you improve database performance? In the early days, it was easy just to move toward host computers that ran on increasingly more powerful processors. Faster disk drives and memory chips helped out also. As memory prices decreased, it became feasible to put more information in memory, which enabled rapid access and update. Most of the current Oracle architecture is designed to maximize the use of memory to obtain speed. Developers also looked at a variety of execution algorithms that could work better in certain circumstances.

The question now is where do you go from here? Although chip makers continue to improve their performance somewhat, they have to work very hard to get a fifty percent increase in performance over their current chips. That is not enough to keep up with user demand, which is increasing at a very rapid rate. The solution that most vendors of servers and host computers have adopted is the concept of putting more CPUs to work in a single computer.

This concept is referred to as *multiprocessing*. There are different types of multiprocessors, but the one DBAs are interested in is the *symmetric multiprocessor (SMP)*. With the SMP, all the processors are just as capable as one another of getting the job done. They all have full access to memory and disk resources. In effect, you have multiple computers that work together as if they were one. This poses some problems for the operating system vendors because they have to find ways to share the load to keep all these processors equally busy. Application vendors have to figure out ways to let the applications take advantage of this architecture by splitting the tasks that need to be performed into smaller subtasks that are commonly referred to as *threads*.

Oracle has spent a lot of time and money in implementing as much parallel processing as possible. It started with multiple database writers in version 7.0 and continued with parallel query and parallel recovery in version 7.1. These are the types of Oracle activities that are most likely to cause bottlenecks (it is unlikely that the checkpoint process will cause you problems).

In addition to parallelizing demanding processes within a single instance, Oracle enables you to have multiple instances accessing the same database. This is the parallel processing option that enables you to have multiple complete sets of background processes servicing a single database. A similar concept can be applied to distributed databases wherein you have multiple instances at multiple locations working on synchronized copies of the data. This not only enables you to have multiple sets of Oracle background processes to meet processing demands, you can actually have multiple computers working on them.

This chapter goes over the basic ways that Oracle is currently taking advantage of parallel processing. This trend should continue. Most of the big database makers are moving toward “parallel everything” in their architectures. Most of the major

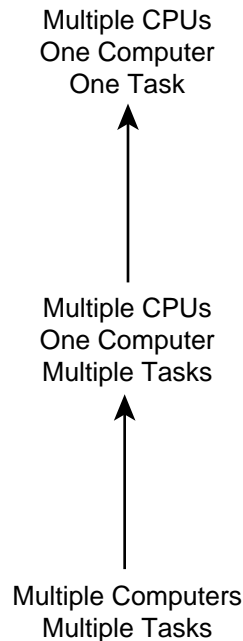
hardware vendors are implementing parallel processors in their computers, and there are computers that are showing up with hundreds or even thousands of processors connected. Therefore, you should consider these parallel processing options and see how you might be able to use them to benefit your particular applications and instances.

TYPES OF PARALLEL PROCESSING

For purposes of this discussion, the world of parallel processing can be broken down into three general categories (see Figure 44.1):

- ◆ Using multiple computers to perform different tasks
- ◆ Using multiple processors within a given computer to service multiple tasks
- ◆ Using multiple processors within a given computer to service a single task

Figure 44.1.
Types of parallel processing.



Oracle has positioned its products to take advantage of all three forms of parallel processing. The distributed database option can be used to have multiple computers (in multiple locations) working on the same set of data at the same time. Multiple processors can be put to work on multiple tasks via the multi-threaded server, parallel processing option, and asynchronous database writer options. Finally,

multiple processors can be put to work on the same task in the parallel query and parallel recovery functions.

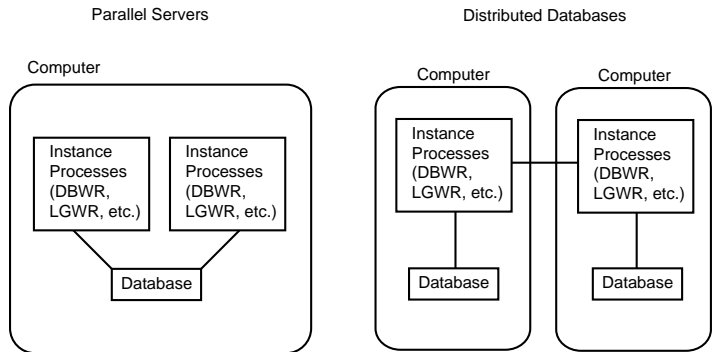
WHEN TO USE PARALLEL PROCESSING

Parallel processing is used mostly on computers that have multiple CPUs. Your goal here is to correct situations wherein you have jobs that are heavily loading a few of the system's processors while the remaining processors sit relatively idle. A classic example of this is a very large query that runs for several minutes on a system that, overall, is lightly loaded. Another use might be a system wherein the database writer process is having difficulty keeping up with the transaction volume (that is, you see a large amount of run time for the DBWR process). You probably will not receive a definite message that tells you to implement parallel processing. Instead, you probably will be searching to improve performance and have to experiment with implementing parallel processing options until you find the ones that help.

DISTRIBUTED DATABASES VERSUS PARALLEL SERVERS

It is important to understand the difference between using parallel Oracle servers and distributed databases. Figure 44.2 shows the differences between these Oracle options. The parallel server gets multiple Oracle instances and sets of background processes (DBWR, LGWR, and so forth) to perform the work needed while accessing a single database. This can be useful when you have a database in which the background processes are overloaded while the CPU complex still has capacity available. The need for parallel servers may well decrease because Oracle continues to implement such parallel processing for its key processes as it currently allows for the database writer and recoverer processes.

Figure 44.2.
Parallel servers versus distributed databases.

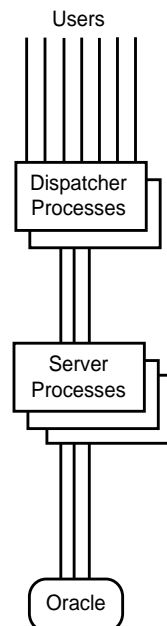


The distributed database option makes multiple copies of the data, usually on different computers. This has the effect of enabling additional processors to work on the data processing tasks. It also enables you to implement different security environments because you have separate databases with separate user and security tables. This function depends much more on your network capabilities to keep up with processing demands. A final consideration is that there is a time delay between when a transaction is committed on one system and when the changes are replicated in all the distributed databases. You can set up parameters to specify how quickly the changes are distributed, but it is still a much slower process than when you are committing changes that are in memory in your local instance. This option is not available on all Oracle platforms (that is, it was not available on the Workgroup Server products when this book was written).

MULTI-THREADED SERVERS

Some might consider the multi-threaded server (MTS) to be a parallel processing option. It is true that you usually have multiple server processes and can even have multiple dispatcher processes, but the multi-threaded server is designed primarily to save memory space when you have a large number of connections on a system (see Figure 44.3). It is true that Oracle could have implemented a single connection server and some form of queue concept to accomplish this goal of saving connection process memory, but that would have really degraded performance. The MTS can therefore be thought of as a way to utilize multiple parallel connection processes to implement shared connection services.

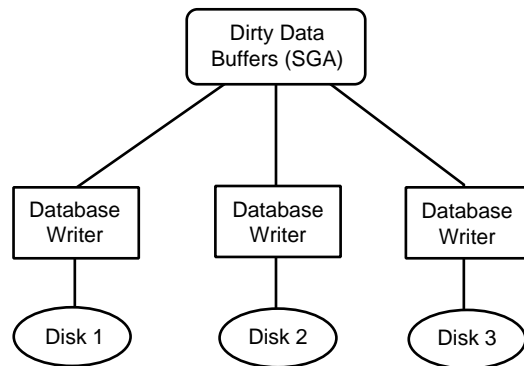
Figure 44.3.
The multi-threaded
server.



ASYNCHRONOUS DATABASE WRITERS

What is the biggest problem with writing data to disks? It is the fact that disks are usually orders of magnitude (powers of 10) slower than the CPUs and memory areas to which they are attached. Remember that famous waiting for input/output (%wio under UNIX) state that you explored in Part 6? That is what you have to deal with when you are trying to perform mass updates to the database. This can become an important bottleneck in that Oracle will *not*, under any circumstances, allow you to overwrite data updates until they have been transferred to disk. You have to love the tenacity with which Oracle enforces its data integrity features (except, of course, when you spend hours applying archive log files to your database that you know are not needed to recover the temporary tablespace data file that you lost due to a disk crash). Any time you see such a bottleneck in your system, it is a good opportunity to implement parallel processing, which Oracle has provided with its asynchronous database writer processes (see Figure 44.4).

Figure 44.4.
Asynchronous database writers.

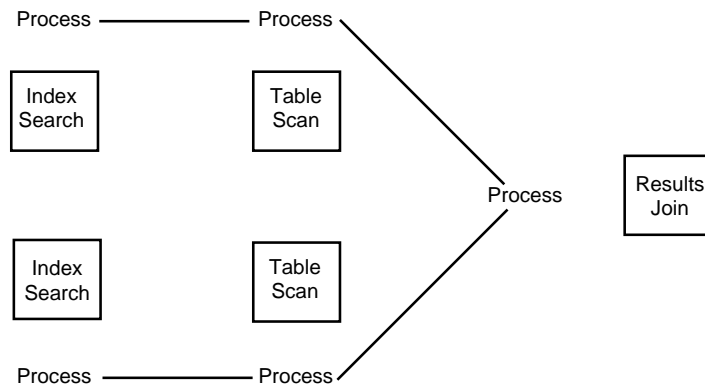


This is actually a very good idea. Your database buffer cache in memory is probably full of updated rows that can be sent to multiple disk drives. Why not have a series of processes divide the overall job into a series of tasks? You have to coordinate to ensure that you do not have a number of processes all competing to write to the same block on disk. However, especially in large databases, the odds are that you can perform this division of labor and break up the load to increase overall input/output performance. This is much simpler to implement than the parallel server option, which gives you multiple copies of all the background processes, but also requires coordination of updates and locks between two SGAs (and therefore you lose the speed of having to deal with only one set of locks and dirty buffers that are cached in memory).

PARALLEL QUERY

As time went along, data warehouses were implemented. They benefited from having multiple database writer processes and some of the other parallel processing features that you have learned about so far. However, they still relied on a single processor and task to perform one of the operations that users are most sensitive to—executing their queries. The problem was that when you implemented a large query that gathered information from a number of tables and combined them, you could use only a single processor to perform all that work. The rest of the processor complex could be sitting idle while your processor chugged away on your query. The parallel query option, implemented in Oracle 7.1, was the answer to this problem (see Figure 44.5).

Figure 44.5.
Parallel queries.



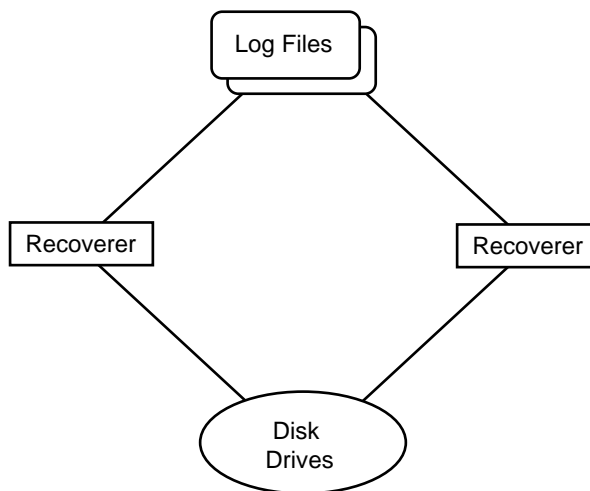
If you think back to the discussions of execution plans, you will find that complex queries (ones that join multiple tables together or use a number of indexes) actually consist of a number of different tasks. Some of these tasks can run parallel with one another (the table scans that are used as input to a UNION ALL query). Others can run in sequence with one another, as in the case of an index search that is used to feed row IDs to a table scan. The parallel query option is designed to assign these individual tasks to multiple CPUs and coordinate their results to produce the desired answer.

PARALLEL RECOVERY

Finally, the parallel processing option that you hope you never have to use is the parallel recovery feature that is implemented in Oracle 7.1. When you perform a recovery, you are in a situation in which Oracle is uncertain about the status of data in the data files. Therefore, it insists on reading through every online redo log and

archive log file that has been created since the time your backup was taken. This can be an extremely time-consuming process. It took days to perform recoveries on one large data warehouse that I worked on where backups were taken weekly and millions of rows of data were applied in a given day (most of the time was consumed swapping tapes for the archive log files). The opportunity exists to have multiple recovery processes working on these logs and applying the transactions as needed to recover the instance (see Figure 44.6).

Figure 44.6.
Parallel recovery
processes.



SUMMARY

This chapter presents an overview of the various parallel processing options that are available within Oracle. These design features are intended to take advantage of the trend toward implementing multiple CPUs on most computer systems that are intended for use as servers (rather than the older implementation in which you tried to make a single CPU more powerful). There are a number of common operations in normal database processing that tend to lend themselves to parallel operations. Examples of this include queries, recoveries, and functions, such as writing modified rows from memory to disk.

It seems reasonable to conclude that Oracle, and other database vendors, will continue their trend toward parallelizing almost every function in their system that can benefit from it. It is much simpler than using earlier options, such as the Oracle parallel server, and has speed advantages because coordination of tasks is done within a single memory area (SGA). One trend that may grow in popularity is the distributed database. Here, the processing occurs on different computers that work to distribute database changes to all other computers that share a given table.

- 
- Software Stored in the Database
 - The Object-Oriented World
 - Trigger Types and Uses
 - Database Procedures
 - Database Packages
 - When to Use Packages, Procedures, and Triggers
 - Summary

CHAPTER 45

Packages, Procedures, and Triggers

Most of the software that is currently associated with Oracle databases is stored in the traditional manner—separate disk files. However, there may be cases where you want to store the software in a nice, controlled location such as the database. With Oracle7, you have the option of storing the software within the database by using constructs such as stored procedures, functions, and triggers. This feature not only provides for a single, controlled place to store the software, it can also improve performance because the queries are stored in both source and parsed formats.

This chapter provides an overview of procedures (including functions) and triggers stored within the Oracle instance. This is not a programming lesson. Instead, the chapter presents sufficient material so that you can understand what is going on with these database objects. You will also explore some of the storage and maintenance ramifications associated with software stored in the database. Along the way, you will look at a basic introduction to object-oriented data constructs that may be useful as a background to see where Oracle is headed (supposedly in the RDBMS version 8 release).

SOFTWARE STORED IN THE DATABASE

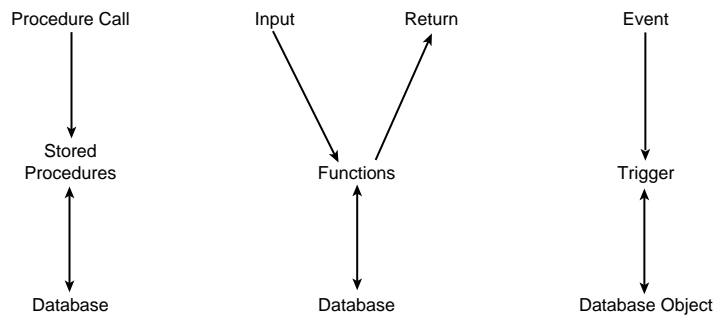
Starting with Oracle version 7, you were enabled to store procedures, functions, and triggers in the database. There were some exceptions, such as Personal Oracle7 for Windows, which does not support triggers in the database (this would be hard in an environment that does not support multiple tasking). Most object-oriented devotees would scoff at this arrangement as being hardly object-oriented. You cannot use the class associations described previously and most of the other constructs of pure object-oriented philosophy. However, it is a start and can be useful to developers and DBAs alike.

Oracle supports three basic types of software objects within the database. The first of these is the *stored procedure*. This is a complete program that is written in SQL and PL/SQL. It typically performs a series of database updates, deletions, or insertions (because PL/SQL is ill-suited to producing reports or other functions). You might consider using a stored procedure to wake up occasionally (start from a cron script) and gather database statistics and store them in a table for later retrieval.

Closely related to the concept of a stored procedure is a *function*. Like a stored procedure, it is a complete routine that is written in SQL and PL/SQL. The key difference is that functions are designed to take a series of parameters as input and return an output value to the users. Functions are typically called by other functions or procedures to perform a repetitive task, such as cross-referencing a value in an input table, performing some calculations on it, and then returning an output value to a number of other software routines.

The final major type of stored object within Oracle is the *trigger*. The concept of triggers comes from many of the Oracle software development tools, such as Oracle Forms. Triggers relate to event-driven computer processing. In the event-driven world, the computer displays some information to the user and then sits and waits for the user to do something (click a mouse, hit a key, or so on). The action taken by the user is referred to as an event. Triggers are the computer's response to the event that is programmed by the application developer. In Oracle, there are a number of events against which you can write triggers. For example, you may want to perform a series of validations on data before inserts occur on a certain table. Figure 45.1 illustrates the differences between stored procedures, functions, and triggers.

Figure 45.1.
Stored procedures,
functions, and triggers.



There are a couple of points that need to be made about these software objects that are stored in your database. The first key point is that they are stored in your `SYSTEM` tablespace. (This really bugs me because I like to keep user stuff out of the `SYSTEM` tablespace.) You can have problems when this tablespace becomes too full. However, that is the way it was implemented and you just have to live with it. It is also important to remember that Oracle stores the procedures (not triggers) in both source code and parsed (ready-to-execute) forms.

Tip

Because Oracle stores the procedures (not triggers) in both source code and parsed (ready-to-execute) forms, you do not have to spend time parsing queries and transactions that are stored in the database already.

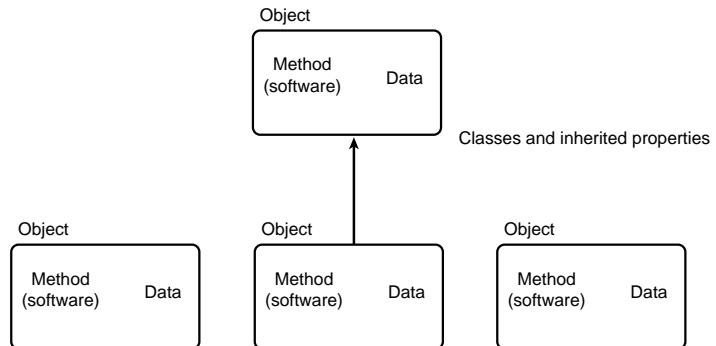
One other construct that you need to consider is the *package*. A package is a collection of stored procedures that are grouped together. You can think of it as a program that is composed of a number of subroutines. The advantage here is that you can call the individual procedures within the package as needed; you do not have to run the entire package. In that respect, it is somewhat like a library of software

routines. The key difference is that the software is stored within the database in a pre-parsed form that is subject to all the database security mechanisms.

THE OBJECT-ORIENTED WORLD

The concept of storing software and data together has been kicked around in the object-oriented world for several years. Most object-oriented languages such as C++ provide some form of associating actions (software) with the underlying data (tables). With concepts that enable you to use common procedures in multiple locations and group objects into hierarchies that share common properties, you can build up a hierarchy of data and actions that model the function of a system fairly well. Figure 45.2 shows some of these basic concepts.

Figure 45.2.
Basic object-oriented concepts.



There are a lot of constructs within the object-oriented world that are designed to increase development productivity and data integrity. However, Oracle does not currently implement many of them. It's a stretch, but you could say that triggers represent a linkage of methods (actions) to data. Views permit some data hiding, and you can overload names (for example, the name in your schema versus that same name accessed through a public synonym). However, most people classify the current versions of Oracle (7.1 and 7.2) as not being object-oriented. For now, all you have to worry about from an object point of view is the concept of storing software within the database and associating some activities with certain database data objects (for example, triggers associated with a given table). The rumors for Oracle 8 say that it will incorporate a number of additional object-oriented concepts.

TRIGGER TYPES AND USES

As stated earlier, triggers are bits of software that are activated when the user performs certain activities. They have a number of uses, including the following:

- ◆ Calculating derived column values from other columns and tables
- ◆ Enforcing complex security and data integrity rules that are beyond common SQL data manipulation statements
- ◆ Providing complex auditing and event logging functions that are not part of the basic Oracle auditing capabilities
- ◆ Centralizing software routines associated with given objects to ensure that they are applied uniformly by all user applications

The trigger is merely a PL/SQL block that performs a series of activities. It can actually contain some fairly complex software. Its main limitation is that the output is typically to other data files rather than the terminal. This trigger is designed to fire after a row is affected or when a statement is executed. Here are the types of triggers that are provided by Oracle:

- ◆ BEFORE statement trigger
- ◆ BEFORE row trigger
- ◆ AFTER statement trigger
- ◆ AFTER row trigger

You can also form triggers that effect multiple activities. For example, the following is a possible trigger:

```
create trigger golf_validate before update or delete or insert on golf_scores
declare
...
end;
/
```

This trigger activates before any data manipulation statement is executed against the `golf_scores` table. Perhaps you want to use it to validate access to the table through some complex scheme (for example, allow updates only between 2 a.m. and 5 a.m. on Sundays). Triggers can be useful when you want to enforce some restrictions or perform routine actions and you do not want to have to rewrite this software in dozens of different applications.

DATABASE PROCEDURES

Stored procedures are relatively easy once you understand the concept of triggers. They are software routines written in SQL and PL/SQL that are not associated with any event. Instead, you have to make a call to them to execute them. This is not so bad. These routines are typically used to store processing functions centrally (not oriented toward data output to users) that are used by multiple routines. The advantage of this construct is that if you want to change the processing routine, you update it in one spot and all the calling routines can take advantage of it automatically. This can be important in organizations in which software maintenance is a major portion of the computer budget.

Another interesting feature is that stored procedures (triggers, packages, and so forth) are fully integrated into the Oracle security scheme. The owner of a stored procedure has to grant execute permission on that procedure to the individual database users or roles. However, there is a little trick that can be used for enhancing security (it also can be a security hole if you are not aware of it). When users execute a stored procedure created by another user, the users executing the procedure access data with all the rights and privileges of the user who created the stored procedure, not with their own rights and privileges. This can be used to write controlled update procedures that allow users to perform updates, but only through the controlled procedures (that is, the owner of the stored procedure has update privileges, but not the individual user). They cannot use their own accounts to make updates from the SQL*Plus prompt. Keep this in mind when setting up your security scheme.

Let's look at an example of creating a database procedure. This routine applies a 7% sales tax to the orders that come from Allegheny County (no, I am not fond of our local sales tax). It might be run at the end of an order entry routine along with a series of other similar routines for all the counties (and perhaps states) to which you ship orders. For now, let's keep it simple so that you can concentrate on the idea of stored procedures:

```
SQL> create procedure taxer
  2  as begin
  3    update orders
  4    set tax = 0.07 * sub_total,
  5    total_order = 1.07 * sub_total
  6    where county = 'ALLEG';
  7  end;
  8
SQL> /
```

Procedure created.

Your next question might be “How do I execute a stored procedure?” It is actually fairly easy. The following command executes this procedure:

```
SQL> exec taxer;
```

PL/SQL procedure successfully completed.

This results in the orders table being updated to look like the following:

```
SQL> select * from orders;
```

CUSTO	DATE_ORDE	SUB_TOTAL	TAX	TOTAL_ORDER	DATE_SHIP	COUNTY
ABC	01-JAN-95	123.34	8.63	131.97	04-JAN-95	ALLEG

Suppose that your first stored procedure is not exactly perfect. You get an error message that tells you something is wrong, but does not tell you what. You can find the details of what was wrong with your procedure by using the `user_errors` view within Oracle (you knew they would have to use a database table to store errors, didn't you?) An example of this might be the following:

```
SQL> list
 1 create procedure taxer2
 2 as begin
 3 update orders
 4 set tax = 0.07*sub_total,
 5 total_order = 1.07*sub_total
 6 where customer = 'MSFT';
 7* end
SQL> /
```

Warning: Procedure created with compilation errors.

```
SQL> select * from user_errors;
```

NAME	TYPE	SEQUENCE	LINE	POSITION
TAXER2	PROCEDURE	1	8	0

PLS-00103: Encountered the symbol "END" when expecting one of the following:
 ; <an identifier> <a double-quoted string>
 <a single-quoted SQL string>
 ; was inserted before "END" to continue.

All this boils down to the fact that I forgot to put the semicolon after the `END` statement on line 7. You also should note that it usually creates the stored procedure "with errors." This means that you cannot use it, but it still exists. Therefore, you need to drop the stored procedure (`drop procedure taxer2`) before you can insert your new, corrected version. An alternative is to use another format of the `create procedure` command (the `create or replace procedure` format) that creates the procedure if it does not exist or replaces an existing one. The format of this looks something like the following:

```
SQL> create or replace procedure taxer2
 2 as begin
 3 update orders
 4 set tax = 0.07 * sub_total,
 5 total_order = 1.07 * sub_total
```

```
6 where customer = 'MSFT';
7 end;
8
SQL> /
```

Procedure created.

DATABASE PACKAGES

Packages are relatively simple concepts once you understand stored procedures. A package is merely a group of stored procedures and functions that are located next to one another. You can call individual procedures and functions from within the package. Think of packages as a nice way to group related software objects. This is similar to software libraries used in other systems.

The first thing to remember is that you create the package in one step and then its body in another. That is the way Oracle made it so you are going to have to live with it. The following is an example of creating all the tax packages listed previously in a stored procedure:

```
SQL> create package all_tax as
2 procedure taxer;
3 procedure taxer2;
4 end;
5
SQL> /
```

Package created.

```
SQL> create package body all_tax as
2 procedure taxer
3 as begin
4 update orders
5 set tax = 0.07 * sub_total,
6 total_order = 1.07 * sub_total
7 where county = 'ALLEG';
8 update orders
9 set tax = 0.06 * sub_total,
10 total_order = 1.06 * sub_total
11 where county = 'BUTLER';
12 end;
13 procedure taxer2
14 as begin
15 update orders
16 set tax = 0.07 * sub_total,
17 total_order = 1.07 * sub_total
18 where customer = 'MSFT';
19 end;
20 end all_tax;
21
SQL> /
```

Package body created.

To execute a procedure within the previous package, you use the following:

```
SQL> exec all_tax.taxer;
```

PL/SQL procedure successfully completed.

WHEN TO USE PACKAGES, PROCEDURES, AND TRIGGERS

There are two good reasons for using these database software constructs—sharing and control. Sharing is a nice idea. You can guarantee that everyone is using the same module when you store it in a common place such as the database. You can also greatly reduce the burden of making changes in a large application or across multiple applications that perform common processing.

The other key reason to use stored software is control. This takes multiple forms. The most obvious one is that someone is assigned as the owner of the software and that users can control, through the use of grants, who is allowed to modify or even execute the software. There is a trickier level to this control, though. When you execute a stored procedure, you execute it as the owner of the stored procedure, not as yourself. Therefore, when users grant you execute permission on stored procedures, you get the right to act as they would with data tables and other objects. This can be used to allow users access to data that they would not normally be permitted to access, but such access would be only through a stored procedure that implements special controls (that is, it enables you to change only certain columns within the table). This can be a very useful trick to keep around in certain situations in which you want to avoid implementing security in a large number of application software modules that are stored on local disk drives.

You should remember one final point about stored procedures, triggers, and functions. When you create them, you may get errors. The problem is that you do not see a useful error on the screen. It merely tells you that there is a problem. To find out what the problem is, you have to query the `user_errors` view. Yes, it is a pain in the neck, but that's the way it works.

SUMMARY

This chapter presented a quick overview of stored procedures, triggers, and functions within Oracle. You are probably not ready, after reading this, to become an expert stored procedure developer. However, this is an overview of what these objects are, their storage implications, and a few tricks as to when you might consider using them. This is enough to function as a DBA for a database that uses stored software routines. If you want additional information, the application developers guide covers this material best; the server concepts manual also has a reasonable discussion on the subject.

- 
- What Is Client-Server?
 - Typical Client-Server Architectures
 - Distributed Databases
 - Summary

CHAPTER 46

Client-Server and Networking

This is the last really technical chapter in the book. It cleans up a few loose ends, summarizes, and gives one last shot at DBA philosophy. The next two chapters provide an overview of where to go from here and then an overview of some of the new products that have come out on the market while this book was in the production process. The good news is that I have saved one of my favorite technical topics for last. I like client-server computing. Sure, some of the applications and products can be frustrating, but I love the idea of separating things into neat little components and of everyone having a specific job to do. I was also the one who sorted his clothes cupboard by work shirts, casual shirts, workpants, and so forth.

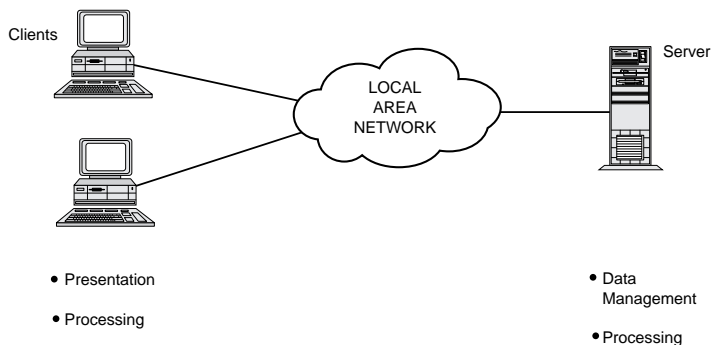
This chapter provides you with a better understanding of this technology, which is where most of the computer environments that I have worked in are moving. You will not be an expert in the packet structures of all of the various protocols, but you should know the basics of how things link and what you need to check on when putting a client-server environment together.

An important topic for those who work in client-server environments is the little tricks that it takes to deal with the unique problems that you run across in an Oracle client-server database. They are actually no worse than the problems that you deal with for host-based applications. It is just that many people have not been in this environment before and therefore need a little experience to make the transition easier.

WHAT IS CLIENT-SERVER?

Perhaps it is easiest to start with a graphic illustration (see Figure 46.1). The concept is simple when looked at from a high level. You have two intelligent devices (your desktop PC and the server) that share the work load between them. This definition can actually be extended to include more than one server, often referred to as a three-tier client-server environment. In this case, you typically have a client, an application server (which handles most of the application processing), and a database server.

Figure 46.1.
Client-server architecture.



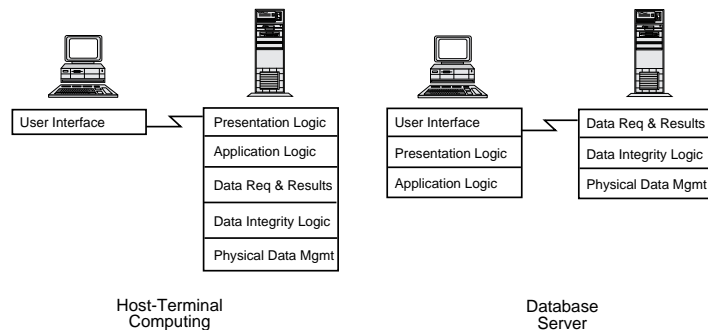
Much of the push for client-server computing has come from the desktop rather than the server. Businesses like applications that have easy-to-use graphical interfaces. It minimizes learning requirements and enables the user to get the job done faster. The software vendors have realized this and have invested a lot in desktop development environments that can provide this kind of interface for the users. Because the desktop is usually too small to store and process large amounts of data, the UNIX server has taken up the role of the central data repository for the data needed by these client-server applications.

One of the variables that exists in the different variations of client-server architectures is the division of labor between the various computer platforms. For purposes of this discussion, let's break up the tasks that need to be performed by an application into six different categories:

- ◆ User interface—what the user sees
- ◆ Presentation logic—what happens when the user interacts with the display
- ◆ Application logic—what business functions are performed
- ◆ Data requests and results processing—what data is needed and what the results are to the request
- ◆ Data integrity logic—what data manipulations are allowed
- ◆ Physical data management—what reads and writes to the storage media should be performed

There are a number of different architectures being sold by the vendors in the industry today. Each one adopts its philosophy based on a judgment of what will be needed, what will sell, and what can it make. Different vendors also tend to go after different portions of the overall market. Transaction processing applications have different needs from those of decision support systems. Figure 46.2 illustrates some examples of this distribution of labor.

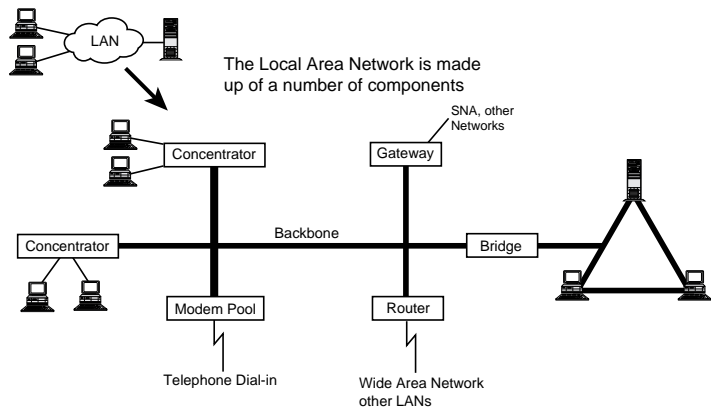
Figure 46.2.
Examples of distribution of labor.



Networks are one of the components that you now have to be concerned with in client-server architectures. In older environments, wherein the data transfer requirements were relatively low between the host and its terminals, very few were concerned with the configuration of the communications system. Today however, as Sun Microcomputer said many years ago, the network is the computer. In many of the cases wherein there have been difficulties implementing client-server environments, it was because the network was not sufficiently developed to support the new applications.

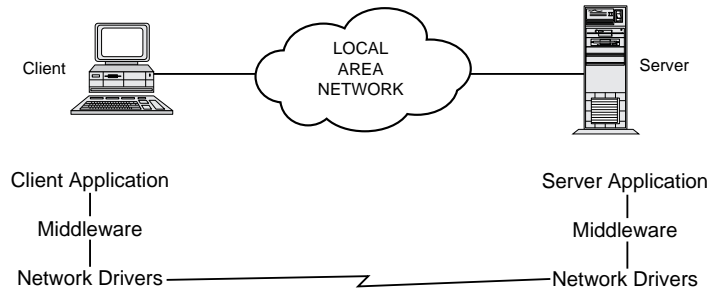
Networks can be confusing things for those who have not worked with them very much (see Figure 46.3). Basically, all you are doing is merging signals so that people can communicate. When you are working on a local network, the experts use terms such as *concentrators*. When you are connecting to remote computer networks via data communications lines, they usually refer to these connections as using *routers*. Finally, if you are going to a network that does not speak the same language as the one you are using, they refer to it as a *gateway*. The key component for most DBAs in the network architectures is the *bridge*. Think of it as a filter. It blocks signals on other networks that do not need to be transmitted to your network (a guy on the third floor printing a large spreadsheet to his network printer). By preventing unnecessary traffic from filtering onto your network, you ensure that you have more capacity available for things such as your new client-server applications.

Figure 46.3.
Basic devices on a
computer network.



The next important concept that the DBA has to master is the question of what software is required to make an application talk with the database (see Figure 46.4). You have to start with the components that you know about (for example, you have chosen your compiler on the PC and Oracle as the database on a Hewlett-Packard UNIX server) and then figure out all the pieces that are needed to connect them. You need to be very demanding and precise in this analysis. It is not good enough for a vendor to say that his product interfaces with PC-based TCP/IP products; you need to know which products and what versions. You have to look hard to ensure that you do not find any strange little “gotchas” that may be lurking.

Figure 46.4.
Communications
software components.



One of the many new terms that you will see in the client-server world is *middleware*. I have seen middleware described as any component that interfaces an application to the network. A more precise definition is that middleware is a type of software that provides a standard socket with which application developers can interface with one another. The key that you have to remember is that middleware is the key enabling technology for client-server computing. You need to ensure that you have all the necessary components and that they work together before you can start building your applications.

Here are the benefits of client-server computing:

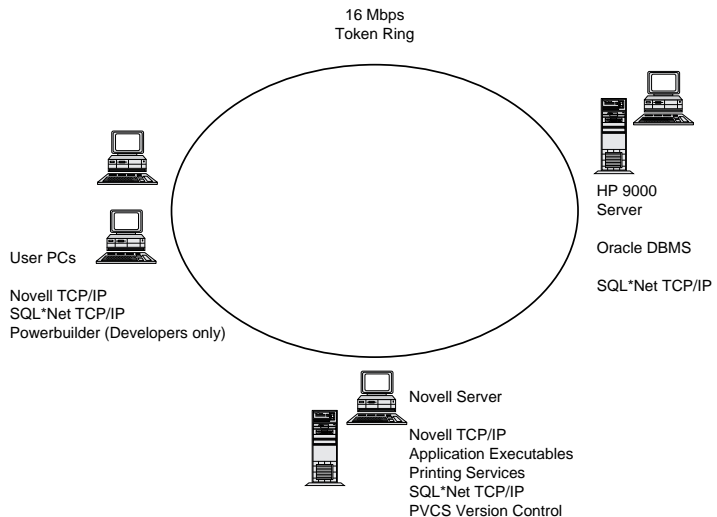
- ◆ *Strength in numbers.* Although the processing capacity of a PC is small relative to that of a mainframe, there are a large number of PCs installed. Added together, the processing capacity of the PCs is often more than that of the central computers.
- ◆ *Ability to choose the right set of computers.* The productivity of a PC graphics user interface can be combined with the storage and compute capacity of servers to produce a more effective computer system.
- ◆ *Cost savings.* The components in smaller servers are much less expensive than those in mainframe computers. By linking the power of PCs and a number of servers, a more functional system can be provided at lower costs.
- ◆ *User productivity.* By proving that the presentation functions by using the windows interfaces on personal computers, applications that are more intuitive and productive can be produced.
- ◆ *Developer productivity.* By dividing a complex application into a number of manageable parts, the application is easier to develop. In addition, the new tools provided on the client side are designed to increase developer productivity.
- ◆ *Flexibility.* Because the clients interface to any number of servers, it is easier to add additional servers to meet increased needs. It is also possible to move applications to more powerful servers.

- ◆ *Open systems.* Good client-server toolkits use industry standards to access various hardware and support applications packages (for example, the RBDMS). Therefore, applications can be easily adjusted to work with different server hardware, software vendors, and alternative network technologies.

TYPICAL CLIENT-SERVER ARCHITECTURES

This section discusses three examples of client-server architectures that present a fair picture of the range of client-server applications and technologies. The first system uses a pretty traditional client-server approach to implement a time-tracking system over a token ring network (see Figure 46.5). The applications were developed by using Powerbuilder and they interfaced to an Oracle 7.0 database running on an HP 9000 server. The middleware consisted of Oracle SQL*Net product, version 1. A little extra was thrown into the architecture with some Novell servers. These servers were used to hold the executable software for the end users. This alleviated the need to load the application on every PC in the building. The key interface concern was obtaining the proper Dynamic Link Library (DLL) for the Powersoft bulletin board to interface with the Oracle database.

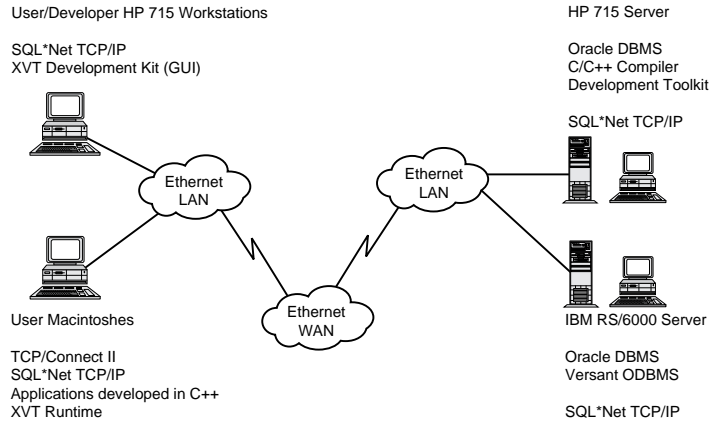
Figure 46.5.
Time-tracking system
configuration.



The next system was an early client-server installation used for an operations scheduling system. This system was actually somewhat of a hybrid between object-oriented and relational technologies, but that is not important to this discussion. In this case, it interfaced the several UNIX workstations and a Macintosh TCP/IP package through SQL*Net to a series of applications that were written in C/C++ and

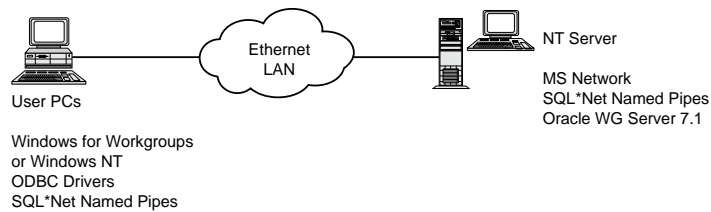
located on a mixture of HP and IBM UNIX servers. The whole thing was far more complex than it needed to be (see Figure 46.6), but it worked, and now it is a good example for people writing books.

Figure 46.6.
Operations scheduling
system.



The final example shows a configuration that can be used in small workgroups (see Figure 46.7). Here, the Oracle is running on a Microsoft Windows NT server running the Oracle workgroups server product. The communications are simplified because the clients (running Microsoft Windows for Workgroups and NT workstation) are already communicating by using the Microsoft network. All that was needed was the Open Database Connectivity (ODBC) driver to be added, and the applications written in C/C++ were ready to talk to the Oracle SQL*Net Named Pipes driver.

Figure 46.7.
Workgroups develop-
ment environment.



TRICKS TO ADMINISTERING A CLIENT-SERVER DATABASE

Administering a client-server database is really not all that bad. There are just a few tricks that you have to keep in mind and everything should go fine:

- ◆ You may run into situations where the user workstation becomes disconnected from Oracle. Unless you have a fairly recent version of SQL*Net 2, this happens when users get tired of waiting for a query to return or finish work for the day and leave Oracle by turning off or rebooting their computers.

The Oracle processes lose track of their communications paths and go crazy trying to reestablish it. I have seen 7 processors on a 12-processor Sequent tied up with these runaway processes (it was ugly). The solution to this is to upgrade to SQL*Net 2 or teach your users that they need to call you when they do this so that you can kill their session from within SQL*DBA.

- ♦ If you are not using the multi-threaded server, you may be tying up a lot of computer memory with dedicated server processes for users who connect in the morning and remain connected until they leave for the day. The solution to this is either to design your applications so that they disconnect after each session (that is, when they get back to a main menu) or implement the multi-threaded server.
- ♦ One trick to improve performance of certain update routines is to store the software within the Oracle database in the form of stored procedures, functions, and so on. If you have update routines that take data from the database, process it, and then store it again without any user interaction or feedback, you have a good candidate for a stored procedure. It saves the load from having to transfer the data across the network for processing and then transfer the updates back again.

DISTRIBUTED DATABASES

Before you leave this subject, you should have a clear picture of the difference between client-server computing and distributed databases. In client-server computing, different portions of the application are processed on different computers. In distributed databases, you have multiple, completely equal databases, each of which is serving a different user community. Your distributed databases may be using client-server, host terminal, or a mixture of these architectures to accomplish their missions.

SUMMARY

This chapter is perfect for a survival guide, but there is much more to know about the subject. This chapter presented an overview of client-server computing and defined some of the key concepts, such as network components and middleware. You learned a few tips for adjusting your administrative plans if you are lucky enough to work in one of these environments.



- The Oracle Environment
- The DBA's Job
- Tips to Make Life Easier
- The Future

CHAPTER 47



Where To Next?

This book emphasizes the breadth of Oracle. You will not need to use every topic, but the wide range supports DBAs from multimedia system developers to basic order-entry system support staff.

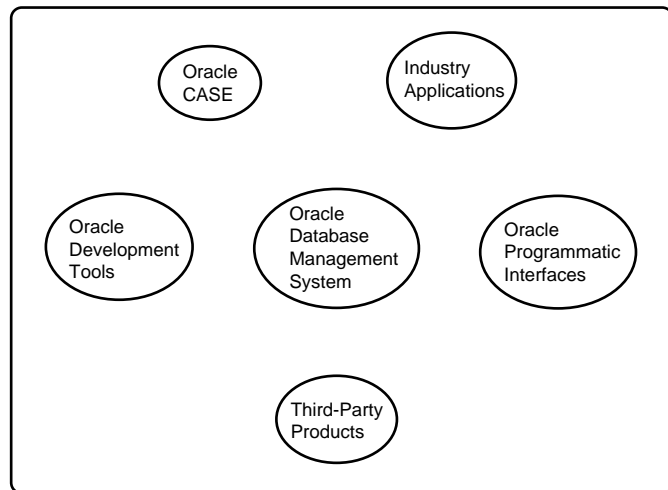
A lot of the book's material relates to building your job description and a system for maintaining your databases and instances. There is still a lot for some of you to learn. If you are working with a data warehouse, for example, you need to take some of the comments about data warehouses and study more advanced topics in tuning and, perhaps, your data warehouse storage schema.

This chapter presents a quick review of some of the highlights and a few words about where you may be going from here. The last chapter in the book is reserved for the product updates and new products that have come out since this book started production. It is easier to add a chapter than to redline everything that is in the editing process. First, you get this review and future directions chapter.

THE ORACLE ENVIRONMENT

Oracle is a corporation. Oracle is a broad range of development and database products (see Figure 47.1). For the DBA, Oracle is mostly a database management system that is entrusted with important corporate data and needs to be maintained by a skilled administrator. This can be a definite challenge as users try to store more information on smaller database servers. These increases usually come at the same time as user expectations for response time become ever more demanding.

Figure 47.1.
The Oracle environment.



Another challenge that separates DBAs from one another is the wide variety of environments in which Oracle is implemented. Operating systems vary from IBM mainframe MVS through Microsoft Windows, with a lot of UNIX servers between these two. These different host environments create unique sets of problems. A problem that you experience may not be faced by your fellow DBAs running on a different operating system.

Oracle also supports a number of different types of database applications. Some DBAs maintain very large data warehouses with limited, batch updates and enormously demanding queries. For others, the challenge is to keep up with the high volume of transactions that an order entry system is placing on the server. These, in effect, become specialty areas for the DBA.

Finally, as if things were not challenging enough, Oracle supports a number of different access methods. The traditional environment, in which users log in to a host computer and access applications that run on the server, still exists. However, there are a number of client server architectures that are popular today. Not only do you have PCs that act as clients accessing a database server, you get clients that access an application server that accesses a database server. Each of these environments presents challenges that you have to rise to as the database administrator.

THE DBA'S JOB

You have to deal with the fact that the role of the DBA varies with different organizations. The job ranges from a full-time position wherein the DBA is responsible only for database concerns of one or more large databases to a part-time position that has to fit with your main career as an engineer or software developer. Create a job description for yourself on which you can get some agreement.

There are a series of tasks that you may be asked to perform as a DBA, and they can often require a juggling act (recall Figure 2.2). There are a wide range of jobs, from installing Oracle to tuning user queries. It is important to control the scope of these tasks to ensure that your job jar is reasonable (and that you get some time at home). Some of the juggling tips in this book may be overly simplified for you. You may want to implement more elaborate schemes. The bare bones suggestions will be useful to the majority of new DBAs who are not working with unusual or demanding Oracle instances.

Proactively manage your database. This can be typified by looking for problems before they become serious rather than waiting for applications to abort. It takes a lot of planning before you actually perform major evolutions such as database software upgrades. Although you can often get away without some of this planning, those times that you get burned by not planning will remind you that the effort spent in planning is worth your time. This is especially important for the part-time or novice DBA who does not have the experience to “wing it” and get away with it.

Planning may be tough because you may not have as much time to spend on planning, but you will have even less time if you have to go through a complex problem repair job on your database.

TIPS TO MAKE LIFE EASIER

Here is a summary of the best tips from this book, my favorite suggestions that I try to follow myself when administering a database:

- ♦ Be proactive whenever possible.
- ♦ Devote a fair amount of time to planning your upgrades and installations of the Oracle software. There are often bugs with the installation process, so ensure that you have enough time to solve problems and always remember to read the README file—it often contradicts the Installation and Configuration Guide.
- ♦ Set up a monitoring program. This is your best chance to catch problems before they become serious.
- ♦ Always expand your knowledge of Oracle and keep up with current developments. Some of these changes can save you a lot of time by solving some nasty problems.
- ♦ Form a network for support. This may begin with an Oracle Technical Support contract, but should extend to the use of resources such as the Internet. Never underestimate the value of having a circle of DBAs to bounce ideas off of and call for help when everything falls apart.
- ♦ Spend some time developing a database administration scheme before things get very far with your database. A little organization can help a lot in managing the chaos associated with the modern business world. It is important to remember to enforce this scheme after you publish it, too.
- ♦ Try to do everything that you can to remain “plugged in” with your developers and end users. It can be difficult when you have a list of tasks to perform that fills up an entire notebook. However, if you keep in touch, you can usually avoid surprises that may cause you headaches in the future (for example, “didn’t we tell you that we were bringing the accounting department into the system tomorrow?”).
- ♦ Finally, do everything you can to get an agreement as to what your responsibilities are. Ensure that everyone understands the level of service that your management is willing to provide (no calls at 2 a.m. for routine questions, for example). It is important to keep you sane and your frustration level low.

THE FUTURE

Now on to your future. Perhaps Oracle database administration will be a career for you. Perhaps it will be something that you do until that executive position opens up for you. This book presented two themes that will help you with the future. The first involves planning and proactive management of your instance—such things as annual capacity planning and routine checking for problems within your instance. The goal here is to help you be ready for the future as it affects your local Oracle world.

The second theme is keeping up with what is happening in the world as it relates to Oracle DBAs. There are a number of resources, such as the Internet, that can be used to keep current with what is happening and what is about to happen. Computers and the Oracle system will continue to evolve at a very rapid pace. The database will be asked to store complex objects (such as blueprints and video programs) in the not too distant future. Hopefully you, as a DBA, will be ready for these changes when they come.

-

Oracle Workgroup Server and Oracle 7.2

One of the problems in writing a book such as this that takes many months of writing, editing, and production is that the development and marketing groups at the vendor are busily introducing new products. When this book began, it included experience with the Personal Oracle7 for Windows Beta release. By the end of the first draft, Oracle made a significant push in the small server market with its Oracle Workgroup Server products *and* brought out its initial releases of the Oracle 7.2 database product.

Rather than pasting my impressions and overviews of these products throughout the book, I thought that it would be easier to add a new chapter to cover these products for those folks who are interested in them. Neither of these products is radically different from the DBA point of view. Actually, the fact that they work pretty much the same as previous Oracle products is one of their most endearing qualities.

This chapter presents the highlights of these products and what they mean to the DBA and developers. It also provides a few tips that I have come across working with the Oracle Workgroups Server under Microsoft Windows NT that may be especially useful for those who have DBA experience under UNIX but are a little lost on these new, smaller servers. Finally, this chapter offers some interesting tidbits about Oracle 8. Of course, the version 8 release of the RDBMS is still many months away and Oracle has the right to change features, but I thought that it might be fun to swap a couple of stories.

FIRST IMPRESSIONS OF THE WORKGROUP SERVER CONCEPT

You read in the trade magazines and fliers from Oracle about this new product push that they are having with their Workgroups products. Perhaps you just toss it aside along with the fliers you get about all those development tools and third-party utilities in your mailbox every day. I have to admit that I read the magazine articles, but never made the time to attend the seminars (after all, I have a UNIX Oracle DBA working on a huge information warehouse system). Next thing I know, I get an assignment to work with a new company in our area supporting its Oracle databases and, you guessed it, it is using that new-fangled Oracle Workgroups Server product.

I had some real technical concerns when I started working with this product. How could a PC (even a fairly powerful Pentium) hope to keep up with the power of a mid-sized UNIX server (I was used to some really big multiprocessor systems). Oracle also is an extremely demanding product in terms of memory and disk input/output. Most PCs have relatively small memory configurations, and their disk subsystems are not as fast as those typically found on UNIX servers.

First, let's start with some of the things that I like about the product. Oracle, like many other vendors, prices its products based on the size of the host computer. That makes the Workgroups products extremely attractive from a price-performance point of view if you do not need a large amount of database processing. (No one cares whether you are using only one percent of the processing power of that large server for database processing; they charge you as if you had the machine running full tilt.)

It is compatible with the other Oracle products, at least as far as the application developers are usually concerned (less distributed database features). This is important when you want to set corporate standards. It also helps to ensure that a developer is productive on a wide variety of server platforms. The Workgroups server is a good choice, in most circumstances, for development and test instances. This forms a nice family of servers that scale well.

Finally, on the good side, it mixes some of the convenient tools that can be found in PC environments with the server products. SQL*DBA and the command-line versions of tools such as Import are just not as friendly as your typical MS Windows application. The tools that you find with the Workgroups server are a mix of the Windows tools and the traditional command-line and menu systems found in other Oracle environments. The GUI interface should enable those who did not grow up with and learn the more difficult tools to administer a database in less time.

Now on to some of the things that I find more difficult to deal with in the Workgroups environment. First, the manuals have not always caught up with the actual products themselves. For instance, the manuals do not mention that you have to make a copy of the initialization file (init.ora) that contains the name of the new instance that you are trying to create before you can create the instance (this is something that the UNIX/Oracle folks are used to). It also fails to mention that when you connect internal, it does not sense your operating system groups. Instead, you have to put in a slash (/) and type the database password. This is something that comes from the Personal Oracle world. Be prepared that you may have to improvise and experiment when things do not work exactly the way the manuals read (remember, they had to start publishing those manuals before the final features were developed).

Next, I find that setting up the communications links is somewhat tougher. Perhaps much of this stems from the fact that I did not have access to the traditional communications utilities (`ftp` and `ping` in UNIX/TCP) that enabled me to test the support components. I had to set it all up and then troubleshoot everything when something did not go right. UNIX types also might have to get used to some of the protocols and utilities that you have to use in the server world (for example, NetBEUI and the network control panel utility).

Another thing that you may have to deal with is the lack of experience in server administration, at least as it relates to setting up a database server. In most large UNIX/Oracle installations, companies see the need to get at least some training or an experienced person to care for the operating system. Workgroups server administrators typically are not afforded these luxuries and do not usually have much experience in using a server for things similar to the Oracle database. Therefore, you may have to spend some time learning and experimenting with the operating system itself.

You also have to realize that you are pushing the limits of these operating systems in many cases. Most server operating systems, such as Novell or NT, have had small databases that operated on the server for several years, but never anything large, such as Oracle. As you install Oracle and other products that are more demanding on these servers, you may start to overload the operating system or server hardware. Perhaps you will never reach this point, but be aware of the potential.

The good news is that the Oracle Workgroups server is so much like the Enterprise edition of Oracle that you do not need substantial retraining on the Oracle side. There are a few new utilities that are easier to deal with after you get used to them, but things are otherwise the same as always. You need to keep an eye on the performance of your server if your application grows. You can exhaust the capacity of a 486 server much more quickly than you could a large Sequent machine. It is an interesting product and very useful. You should keep this product in mind for development and test instances. There are few organizations that can afford Oracle on a larger server that cannot afford to install a PC server with the Workgroups products on which the developers can work. This need alone justifies the existing of the Workgroups product line.

ORACLE 7.2

Oracle 7.2 is merely a maintenance release. There are a few nice, new features, but the world has not changed. As a matter of fact, most people out there will probably not even notice if you upgrade your server from 7.1 to 7.2. That does not mean it is insignificant. You always want improved versions of the products—those that you depend upon especially. The highlights of the improvements list include the following:

- ♦ Index creation is now done directly to the database instead of to the buffer pool. This should improve performance.
- ♦ You have the option of creating and updating indexes without writing data to the online redo log files (why bother, you can always re-create them from the tables if you need to).

- ◆ You can specify more complex subqueries in your SQL `select` statements (for example, put a `select` in the `from` clause).
- ◆ You also get cursor variables that enable you to get the result of a query from a stored procedure.
- ◆ Support for the SNMP (Simple Network Monitoring Protocol), that enables general-purpose monitoring packets to monitor Oracle databases, is offered. Perhaps some day you will get to the point where you can monitor the whole computer environment from a single GUI-based screen.

There are a number of other features, such as additional National Language Support (NLS) character sets and the capability of preventing access to the source code of stored procedures in the database. However, the ones listed are the ones that caught my attention.

RUMORS AND GOSSIP ABOUT ORACLE 8

Based on some of the articles published recently, there seems to be two themes that come up with regard to Oracle 8. The first is that Oracle will try to reduce per-user memory requirements wherever possible to enable Oracle instances to support more users. Several host computers running Oracle cannot install any more memory even if the clients had enough money for it. All the memory expansion slots are filled. Therefore, to grow and support really large systems, Oracle is going to have to get a little leaner. There is some room for improvement in this area. I created a simple C application to interface with Oracle by using the Pro*C precompiler. I merely established a connection with the Oracle database. The executable that was produced took up almost 3M. I tried to take out unnecessary include libraries and so forth from the precompiler, but it would not allow me to do it.

The other feature that Oracle (and most other database vendors) are trying to include is support for object-oriented data. Some of the constructs that are needed include the capabilities of storing user-defined data types (for example, X-ray images) and of tying the data with the processes that are associated with this data. Oracle needs to merge the object and relational worlds and keep an eye on open standards (such as SQL), which tend to evolve slowly. It will be interesting to watch. This technology will be one of the keys to enabling the users to store all their data in the database and not merely the things that fit well into a tabular format.

SUMMARY

This chapter presented a brief overview of the Oracle Workgroups server and Oracle 7.2. Don't worry about the brevity of this chapter—it was intentional. The world is not changing due to these products. One of their most endearing features is that they

work pretty much like the other products that you have come to know so well. The Workgroups server market enables Oracle to support the large number of PC servers that are out there today and also offer a low-cost alternative to buying a mid-size or large server installation. The 7.2 product is primarily designed to fix bugs in previous versions of the product, but it also has a few additional features that can improve performance for those who have more demanding applications. As with all releases of the Oracle software, test them out thoroughly before you move your production applications over to them.

- 
- SQL Commands
 - Glossary of Terms
 - SQL*Plus Features
 - SQL*DBA Features
 - SQL*Loader
 - Import and Export
 - Where to Get More Information
 - Sample System Configuration Analyses
 - The Disk Contents

APPENDIXES

Appendixes



- Object Creation Commands
- Object Modification Commands
- Object Deletion Commands

APPENDIX A



SQL Commands

The SQL Language Reference is far more complete than what is presented here. The quick reference card is also far more convenient. This appendix weeds out all the complicated options and presents the syntax of the common commands used by the DBA, and it is arranged in a simpler pattern. Functions are listed by the type of action and the object upon which is acted. Rather than having to know that you start archive logging by issuing an `alter database archivelog` command, in this appendix, you start archive logging by looking in the section titled *Modifying Archive Log Status*.

OBJECT CREATION COMMANDS

Create Additional Data Files

```
alter tablespace add datafile 'filename' size new-size
```

This command is the basic means for expanding the size of one of your existing tablespaces. The filename should be a fully qualified filename (for example, `/data1/oradata/test/system2.dbf`) rather than one that assumes your current directory. The *new-size* parameter is the number of bytes to allocate to this file. It should be in bytes, followed by a K (kilobytes) or an M (megabytes). Here are some examples:

```
alter tablespace system add datafile '/data1/oradata/test/system2.dbf' size 100M;
```

```
alter tablespace users add datafile '/data1/oradata/prod/users2.dbf' size 10000;
```

Create Additional Tablespaces

```
create tablespace tablespace-name  
datafile 'filename' size new-size
```

This command allows you to make new tablespaces and their associated data files. The filename should be fully qualified (for example, `/data1/oradata/test/system2.dbf`) rather than one that assumes your current directory. The *new-size* parameter is the number of bytes to allocate to this file. It should be in bytes, followed by a K (kilobytes) or an M (megabytes). Here is an example:

```
create tablespace sales_tables  
datafile '/data1/oradata/test/sales_tables1.dbf' size 500M;
```

Create a Table

```
create table [schema.]table-name  
(column 1      type(size),  
...)  
mintrans #  
maxtrans #
```

```
tablespace tablespace-name
storage (initial #
        next #
        pctincrease #
        minextents #
        maxextents #);
```

This command creates a new empty table in the specified tablespace. As a DBA, you can typically create objects in other users' schemas. You should specify all the parameters unless you are absolutely certain that you want to accept the default parameters. The following is an example of this command:

```
create table golf_scores
mintrans 1
maxtrans 100
tablespace users
storage (initial 100K
        next 100K
        pctincrease 0
        minextents 1
        maxextents 200);
```

Create an Index

```
create index [schema.]index-name
on [schema.]table-name (column1, column2, ...)
mintrans #
maxtrans #
tablespace tablespace-name
storage (initial #
        next #
        pctincrease #
        minextents #
        maxextents #);
```

This command creates an index containing the specified columns of the specified table. You should specify all the parameters unless you are absolutely certain that you want to accept the default parameters. The following is an example of this command:

```
create index golf_score_people
on golf_scores (golfer_id)
mintrans 2
maxtrans 100
tablespace user_indexes
storage (initial 50K
        next 50K
        pctincrease 0
        minextents 1
        maxextents 200);
```

OBJECT MODIFICATION COMMANDS

Modify a Tablespace

```
alter tablespace tablespace-name offline/online;
```

```
alter tablespace tablespace-name
default storage (initial #
                  next #
                  pctincrease #
                  minextents #
                  maxextents $);
```

There are two common forms of this command. The first takes the tablespace offline or places it back online (the data in it is not accessible or it is accessible, respectively). The other typical modification is that you adjust the default storage parameters for objects created in that tablespace. Note that a third common form of the `alter tablespace` command, the one which adds a data file, is covered under the previous section about creating additional datafiles. The following is an example of this command:

```
alter tablespace users offline;
```

Modify a Table

```
alter table [schema.]table-name
storage (next #
          pctincrease #
          minextents #
          maxextents #);

alter table [schema.]table-name
add (column1 format, column2 format,...);

alter table [schema.]table-name
modify (column1 format, column2 format,...);
```

This command allows you to modify most of the storage parameters for tables that already exist within your Oracle instance. You can also use this command to add new columns or modify existing column formats. Some examples include the following:

```
alter table golf_scores
storage (next 200K);

alter table golf_scores
add (mulligans number(9));
```

Modify an Index

```
alter index [schema.]index-name
storage (next #
          pctincrease #
          minextents #
          maxextents #);
```

This command is used to alter the storage parameters of an index. The following is an example of this command:

```
alter index golf_score_people  
storage (maxextents 250);
```

Modifying Archive Log Status

```
alter database archivelog/noarchivelog;
```

This command either starts archive logging or stops it. Note that you need to set up the archive log destination, the archive log file format, and other initialization parameters before you can begin archiving.

Modifying Redo Log Files to be Offline

```
alter redo log log-name offline/online;
```

This command takes a redo log file offline or online. You cannot take the currently active online redo log file offline (see the next command for modifying the active online redo log file).

Modifying the Active Online Redo Log File

```
alter system switch logfile;
```

This command makes the next scheduled redo log file active whether the current log is filled.

Modifying Rollback Segments to be Offline

```
alter rollback segment segment-name offline/online;
```

This command takes a nonactive rollback segment to an offline status where it cannot be used for transactions. It is also used to restore a redo log file to active status. The following are some examples of this command:

```
alter rollback segment r01 offline;
```

```
alter rollback segment r01 online;
```

OBJECT DELETION COMMANDS

Deleting a Table

```
drop table [schema.]table-name
```

This command drops the table that is specified (removing both data and structure). The following is an example of this command:

```
drop table jgreene.golf_scores;
```

Deleting an Index

```
drop index [schema.]table-name
```

This command drops the index that is specified. Note that indexes are automatically dropped when their corresponding tables are dropped. The following is an example of this command:

```
drop index jgreene.golf_scores_people;
```

Deleting a Tablespace

```
drop tablespace tablespace-name including contents;
```

This command drops the tablespaces and database objects that are located within it. If you do not specify the `including contents` option and there are any objects in the tablespace, your command will fail. Note that the tablespace should be (but does not have to be) offline before it is dropped. The following is an example of this command:

```
drop tablespace users including contents;
```

Deleting a Rollback Segment

```
drop rollback segment segment-name;
```

This command drops the rollback segment specified. Note that the rollback segment must be offline before it can be dropped. The following is an example of this command:

```
drop rollback segment r01;
```




APPENDIX B



Glossary of Terms

Alert log A file in the Oracle system that captures major processing events, such as startup, shutdown and log switches, for future reference. This file is usually located under the `bdump` subdirectory of the `admin` subdirectory for a given Oracle instance.

Archive log A file that contains a copy of one of the online redo log files. Because the online redo log files are written to in a cyclical fashion, their record of transactions applied to the Oracle database is lost if they are not copied to one of these archive log files.

Backup The process of making a copy of some or all of the information contained in a database and storing it on a separate medium (for example, tape). This data can be used to restore the database to operation in the event of a loss of data.

CSI number The Oracle customer service ID number. This is the number that identifies you as a customer and a particular database server to Oracle technical support. It is important to have this number (or numbers if you have multiple servers) handy when problems arise.

Database A collection of data and supporting files that stores information in an organized manner.

Data Definition Language (DDL) SQL language commands that are used to define the tables and other objects that store information within the database.

Data Dictionary An internal Oracle structure that captures a description of the objects that are stored within a database. This is the key to having Oracle find information that you store within the database.

Data Manipulation Language (DML) SQL commands that are used to make modifications to the data within the database. These statements include the `insert`, `update`, and `delete` commands.

Disaster recovery The process of recovering from a serious or complete loss of existing data processing capabilities. Examples might include the complete loss of a host computer or damage to the entire data center.

Instance A collection of memory areas and Oracle background processes that serve to connect users with the information contained in an Oracle database. This is the processing portion of what most people refer to as the Oracle database.

Normalization A process of organizing the data in an application into a logical series of tables. These tables are designed to be efficiently processed by a relational database management system such as Oracle.

Online redo log A series of files that are written to in a cyclic fashion and that hold a record of all transactions applied to the database. These files are one of the keys, along with archive log files, to having the database recoverable to the state that it was in just prior to a loss of data.

Oracle Export A utility that is provided with all Oracle databases to enable the user to copy all or part of the information stored within an instance to an operating system or tape file. This file format has remained constant for some time and is constant between releases of Oracle on different operating systems. It can be used to transfer data between Oracle instances and also to move some of the data out of the database while you de-fragment objects or tablespaces.

Oracle Import Provided with all Oracle databases, the complementary utility to Oracle Export. This utility takes the data produced by Export and inserts it into the Oracle database.

Recovery The process of restoring information in an Oracle database to the state that it was in just prior to the time before data files were lost (disk failure).

Rollback segment A database object that is used to hold before-change images of modified records. There are a fixed number of rollback segments that are used in a rotating pattern.

Rollback tablespace The tablespace (most often called RBS) used in most Oracle systems to hold the rollback segments.

SQL*DBA Provided with all Oracle databases, a utility used to perform common database administrator functions, such as starting and stopping the instances. It also provides a number of monitoring routines that can be used to check the activities that are occurring within the Oracle instance. Oracle Server manager will eventually replace SQL*DBA.

SQL*Plus A general-purpose utility, provided with all Oracle databases, that is used to interact with the Oracle data from a command line. It has a number of formatting controls and can be used to run basic reports. It can execute PL/SQL commands. You can call SQL*Plus with the name of a specific script to execute those scripts from the UNIX command prompt automatically, thereby enabling you to run jobs through scheduling utilities such as cron.

System tablespace The heart of the Oracle database. In this tablespace are stored all the data dictionary objects and internal Oracle tables.

Temporary tablespace A tablespace in which Oracle creates certain temporary data structures that cannot fit into available real memory. An example of this is a large sort.

Tools tablespace A tablespace normally created by the installer and used to hold control tables for the Oracle development tools. Typically, this tablespace is empty unless you choose to use Oracle Forms, Oracle Reports, and the other Oracle tools. You can put the tables for these products in any tablespace if you want.

Trace files Files that are created automatically by Oracle. These files are created when you turn on the trace utility for your session (as when you are trying to determine your execution plan). They are also produced automatically when user and Oracle background processes fail with serious errors. They are the key to your determining what happened when the database crashes.

- 
- Calling SQL*Plus
 - Output Formatting
 - Working with SQL Files
 - SQL*Plus Versus SQL*DBA

APPENDIX C

SQL*Plus Features

This appendix is designed to give you a brief overview of the SQL*Plus utility. There are several books in the Oracle documentation set that cover this utility in more detail; this appendix captures several of the key features.

CALLING SQL*PLUS

There are three typical ways to call SQL*Plus. The first way is used when you are at the command line and just want to get access to the command editor. Use something like the following:

```
$ sqlplus jgreene
```

In this case, you are prompted for your password after entering SQL*Plus.

The second way is in scripts that you run from the operating system command line (shell scripts). You need to specify a routine to execute on the command line (and the SQL script needs to have an exit command at the end of it). You also need to either put in the user ID and password or use operating system authentication (an ops\$ account) that is accessed with the / parameter as follows:

```
$ sqlplus / @script_name
```

This can present some security problems. Most places do not like shell scripts that have the passwords hard-coded into them, and you cannot always use ops\$ accounts. One solution to this problem under UNIX is to store the password in a separate file that is protected. Then, only the file owner can read it (400 protection). The software contains the following command line, which redirects the contents of the password file as the standard input to SQL*Plus and therefore fills in the blank when SQL*Plus asks for a password:

```
$ sqlplus username @script_name < password_file
```

OUTPUT FORMATTING

There are a few commands that I routinely issue at the SQL*Plus command line to make the output a little more readable. This section explores some of the most common.

The first command:

```
set pagesize ##
```

enables you to specify the number of lines that are printed or displayed on a page. For printed output, this is typically around 60 lines. For a normal terminal screen,

it is about 24 lines. I adjust this to ensure that the reports come out evenly spaced on printed pages and also use it in conjunction with the `set pause on` command described next to display screen output one screen at a time. (Otherwise, it scrolls by you far faster than you can read it and may be lost if you do not have enough of a screen buffer to capture all the data.)

The next command:

```
set pause on
```

causes SQL*Plus to stop at the end of every page (see the `pagesize` parameter) and wait for you to press Enter. This is convenient when you wish to read your information a screen at a time on long queries.

This command:

```
column column_name format a25
```

is an example of fixing your column format for character output. There are a lot of fields in the database that are sized very large but rarely filled. When Oracle has too many columns to fit on a single line, it starts wrapping the text on multiple lines. This can be more difficult to read; therefore, you can use this command to shorten unnecessarily long fields so that you reduce the number of lines needed to display a row. The example adjusts the column width to 25 characters.

This command:

```
column column_name format 999,999.99
```

is an example of fixing your column format for a number. It displays the number, showing up to six characters to the left of the decimal point (with thousands separated by a comma) and two digits to the right of the decimal point. This can be very useful to a DBA, especially when you issue queries that are retrieving row counts and table sizes in bytes. It is much easier to read, and much less likely that you will make a mistake, if your output reads 12,555,117 rather than 12555117.

This command:

```
spool file_name
```

is the basic means of taking what you see on the screen as the output of your queries and transactions and storing it in data files for later use. Most of the reports on the enclosed disk are based on having this command write out the results. It can also be a useful way to capture data for later transfer to your PC (for example, you want to capture the text of a query and its error messages).

This command:

```
spool off
```

stops the spooling of SQL*Plus output that is going either to a printer or a file.

This command:

```
spool print_queue
```

is similar to the `spool` with a `filename` parameter. The difference here is that the output is sent to an operating system print queue as opposed to a file. The exact format for the print queue name and how this functions varies between different operating systems. You need to see your system administrator and see which print queues have been set up for you.

This command:

```
prompt
```

is a useful construct in scripts. It echoes whatever you type after the keyword `prompt` on the screen or report output. It is the easiest way to insert text. (I used this a lot in the monitoring reports to output the criteria and other text that separated the various sections of the report.)

This command:

```
accept variable format prompt text
```

enables you to take input from within SQL (*not* PL/SQL). You can build a script that performs an operation on a table, such as listing the row and column comments. If you ask the user to input the table name and then reference that variable in your SQL (by typing `&variable`), you can make this script generic and use it for multiple tables.

This command:

```
column column_name noprint
```

is a variation of the column formatting command that enables you to suppress printing of a particular column.

WORKING WITH SQL FILES

Once you are at the SQL*Plus prompt, you can execute SQL files that you have stored on disk. To do this, you merely input the ampersand (`&`) and then the filename.

You may have to use a fully qualified filename if you are not in the same directory as the script. The following is the format for this command:

```
@script_name
```

Note that this command format works well for both pure SQL and PL/SQL routines.

Another important set of commands includes those that you use to change the inputs on a line. Suppose, for example, you list your SQL buffer and notice that you typed something wrong (you misspelled the table name in the next example). To correct this problem, you ensure that you are on line 2 by typing 2 and pressing Enter at the SQL prompt. SQL*Plus will echo line two for your review and return you to the command prompt. To change the word `gold_scores` to `golf_scores`, you use the change command. Enter the letter `c`, then enclose what you are changing from and what you are changing to within a set of slashes as follows:

```
SQL> list
1  select score
2  from gold_scores
3  where golfer = 'JGREENE'
```

```
SQL> 2
2  from gold_scores
```

```
SQL> c/gold/golf
2  from golf_scores
```

```
SQL>
```

Suppose that you have entered a script that you might want to use for future reference. To save the contents of the command buffer, you type the following:


```
SQL> save filename
```

If the file already exists and you wish to overwrite it, type the following:

```
SQL> save filename replace
```

SQL*PLUS VERSUS SQL*DBA

It's a good idea to use SQL*Plus for most of your query and reporting work—putting commas in numbers and things like that. SQL*Plus cannot be used for functions such as starting and stopping the database (you use SQL*DBA or Server Manager for this purpose). It also lacks the monitor utilities that SQL*DBA has (`monitor session`, `monitor locks`). However, for running SQL and PL/SQL (with the procedural option installed in your database), it works well.

- 
- Command Line
Versus Screen Mode
 - Menu Interface
 - Monitors
 - Killing User Sessions
 - Oracle Server
Manager

APPENDIX D

SQL*DBA Features

This appendix provides the highlights of the SQL*DBA utility, including the features that I use most within SQL*DBA and some tips on how I like to use it. The server utility guide provides additional details.

COMMAND LINE VERSUS SCREEN MODE

SQL*DBA can make it easy for you to execute functions from its pull-down menu system. There are two reasons to access SQL*DBA in line mode. First, your terminal type may cause problems with SQL*DBA due to a lack of compatible terminal emulations. While you are working out your problems, you can still use the line mode of SQL*DBA. The second reason to use line mode is to execute scripts with SQL*DBA. SQL*DBA's screen mode does not execute scripts for you. Therefore, you have to specify line mode. To run a script from the UNIX shell prompt with SQL*DBA, you type a command similar to the following:

```
$ sqldb lmode=y <script_name
```

If you type just SQL*DBA, it assumes that you want to use the menu mode (which was tough for those of us who used Oracle 6—the command mode was the only mode). You have to do a few things to make your SQL scripts run under SQL*DBA. First, you have to put a connect internal statement at the beginning of your SQL script. (Note that you can connect as any user ID/password combination, but many DBA scripts need to be run this way so that they can perform shutdowns, startups, and so forth properly.) This utility has to be run by a DBA; therefore, the DBA should have permission to connect internal. The capability of connecting internal is determined by membership in the correct operating system group. See your system administrator to ensure that you are a member of the right group (which is normally called DBA).

MENU INTERFACE

The basic menu system for SQL*DBA is shown in Chapter 5, Figure 5.2. The nice thing about this interface is that you do not have to remember the syntax of the SQL commands to perform common DBA functions. If you grew up in Oracle 6, you probably learned to activate things from the command line and find it quicker to do it the old-fashioned way. Remember, the menus are like the old Lotus for DOS spreadsheet menus. You need to press a key to activate the menu (usually the Ins key on the numeric keypad) and then press letter or arrow keys, followed by the Return key, to choose items. If you have a mouse, it definitely is easier to use the menus (more on this in the discussion of Server Manager).

MONITORS

The best feature of SQL*DBA is the monitoring capabilities that it provides. The following are some of the most useful monitors:

- ◆ `monitor session` shows you who is connected to Oracle and what each is doing.
- ◆ `monitor locks` shows you any locks that are in place or waiting to be put in place to detect lock contention.
- ◆ `monitor fileio` shows you the input/output traffic on an Oracle file-by-file basis.

KILLING USER SESSIONS

One of the more important capabilities for an Oracle DBA, especially in SQL*Net 1.0, is the ability to kill a user process. This is needed when the user disconnects from the network in a non-graceful manner (Ctrl+Alt+Del). The Oracle process may be left hanging. You, as the DBA, need to go into the `monitor session` command and figure out the session ID of the user. From this, you can issue the `kill session` command (use the pull-down menus or Esc+k) to kill that process.

ORACLE SERVER MANAGER

Perhaps you are not totally in love with SQL*DBA. If so, you are not alone. It is a bare-bones utility that allows you to work with only one instance at a time. The product that works under MS Windows and Motif (X Window in UNIX) is the Server Manager Product. This product has a lot of the same types of interface as the Personal Oracle7 for Windows utilities that are described in Chapter 5. If you are having difficulty working with SQL*DBA, you should evaluate this product to see whether it fits into your computer environment and budget (because you will usually get this on an upgrade and you need to have a maintenance contract that supports upgrades).



APPENDIX E



SQL*Loaders

The SQL*Loader utility is used to transfer data from an operating system text file (usually ASCII) into the Oracle system. This is one of the simplest means of getting data from legacy systems into an Oracle database. It is not sophisticated, like some of the network gateway products that are available, but sometimes getting the job done simply and on time beats doing it in a fancy manner but delivering late. SQL*Loader does provide a wide range of options. This appendix highlights some of the features. The Oracle utilities manual covers the wide range of options and exact syntax specifications that you may want to use.

USING LOAD TABLES AND POPULATION SCRIPTS

One of the things that you may have to deal with, especially when coming from older, less-flexible legacy systems, is having to do a fair amount of work on the format and content of the data once it comes over to Oracle. You have two options. The first is to use some of the more advanced features of SQL*Loader to convert character text to dates and other such functions. SQL*Loader even has some basic constructs to enable you to reject rows that should not be imported. However, you have another option. You can merely bring the data into Oracle in whatever format the legacy system puts out. You can put everything into flexible varchar columns in a temporary holding table, which I like to call a “load table.” You can then use the full power of SQL and PL/SQL in scripts to filter, convert, and message the data when moving it from the load table to the permanent storage table. You might also do this in cases wherein you have to see whether the row already exists in the database to determine how you process the new input record (that is, do not overwrite certain fields if this is an update).

FIXED COLUMN VERSUS DELIMITED FORMATS

There are two basic means of telling SQL*Loader what data belongs to what column. The first is to use a format similar to the old flat files wherein you say columns 1–7 contain this field, columns 8–10 contain that field, and so forth. The other option is to use a delimiting character, such as a comma (,), to separate the fields. The character-delimited format is easier to write loaders for, but the legacy system output routine may have trouble getting the format correct. I have always found it easier to get data from mainframes in the fixed-column format.

AN EXAMPLE CONTROL FILE

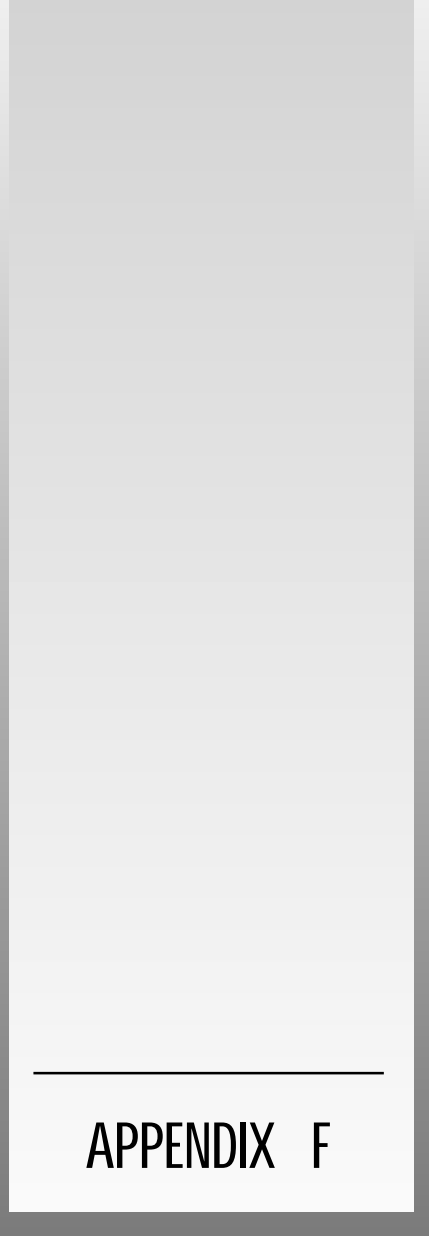
The section in the utilities manual devoted to SQL*Loader is actually quite thick. There are a wide range of options and neat things that you can do with this utility. This example reads data from a column delimited file into an Oracle table:

```
load data
infile '/users/transfer/golf.dat'
insert into table golf_scores
(golfer    POSITION(1:25)   char,
score     POSITION(27:29)  number,
course    POSITION(30:59)  char)
```

USEFUL OPTIONAL PARAMETERS

There are a couple of parameters that you should consider setting up when you run a loader routine. The first is specified in the command line for SQL*Loader. You have the option of adding a parameter in the format `log=filename`. This stores the output of the SQL*Loader session in an operating system text file that you can review later. This can be especially useful when you are loading a long input file and your screen scrolls by with status messages showing the number of rows loaded.

An option that you can specify within the load control file is `badfile filename`. This causes SQL*Loader to store a copy of all rows that it determines are not correctly formatted (for example, a letter in a column that is supposed to contain a number) in a small file for your review. It is usually easier to spot the problem in a small file where everything is bad than to look through a large file where most of the records are good. A similar option is `discardfile filename` within the control file. Rather than incorrectly formatted records, this table stores rows that match the acceptance criteria that you put in for the loader. Again, this can be helpful when you look at the log and wonder why 10 rows were rejected.



Import and Export

The Oracle Import utility enables you to retrieve information that you stored using Oracle Export. This appendix includes a few tricks that I like to use with these utilities.

COMMAND LINE VERSUS INTERACTIVE

You have the option of trying to specify all the controlling parameters that you want to use for an import on the command line or to have Oracle prompt you for answers interactively. Unless you intend to perform the exact same import multiple times, always use the interactive mode. It is not that hard, it does not take that long, and it asks you for all the more important parameters. You might forget parameters or make a syntax mistake if you try to fit it all on the command line.

IMPORT PARAMETERS

The following is a list of parameters that you can specify in a parameter file, at the command line, or interactively when prompted:

- ◆ **USERID** is the Oracle userID/password that will be used to perform the import.
- ◆ **BUFFER** specifies the size of the import data buffer.
- ◆ **FILE** is the name of the input file.
- ◆ **SHOW (Y/N)** lists the contents of the file but does not perform an import.
- ◆ **IGNORE (Y/N)** ignores the creation errors that result if the object already exists.
- ◆ **GRANTS (Y/N)** imports grants.
- ◆ **INDEXES (Y/N)** imports indexes.
- ◆ **ROWS (Y/N)** imports rows.
- ◆ **LOG** creates a copy of the screen output from the import process in the filename specified.
- ◆ **DESTROY (Y/N)** overwrites tablespace data files.
- ◆ **INDEXFILE** writes table/index information to the filename specified.
- ◆ **CHARSET** specifies the character set (for example, ASCII) that is used for the file being imported.
- ◆ **FULL (Y/N)** imports the entire data file.
- ◆ **FROMUSER** is the name of the current owner of the objects.

- ◆ TOUSER is the name of the user who is to own the objects after the import.
- ◆ TABLES lists the table names to be imported.
- ◆ RECORDLENGTH is the length of a record of import.
- ◆ INCTYPE is the type of incremental import.
- ◆ COMMIT (Y/N) commits an array insert.
- ◆ PARFILE is the name of the parameter file containing this type of Oracle Import parameter.

REQUIREMENTS FOR RUNNING IMPORT

You need to ensure that the SQL script CATEXP.SQL, located in the rdbms/admin directory, has been run. You may also need additional privileges, such as create any table, if you are importing objects that were owned by another user.

ORACLE EXPORT

Before leaving this topic, I want to add a few words about Import's complement, the Oracle Export utility. The Import parameters are basically those used for an output. I want to add my vote for using the interactive form of the Export command when performing most exports as a DBA. There are a number of parameters that can be set incorrectly (for example, grants=N) if you use the command-line method and do not specify all the possible parameters. This can cause problems on import because you probably will drop the object before you re-create it. You may then wind up having to do a lot of cleanup work (re-grant access permissions and so forth).

AN EXPORT AND IMPORT EXAMPLE

Perhaps it would be useful to show an example of an export and import of Oracle data. This example takes all the tables in one NT Server instance and transfers them to another. The commands and output for the export go something like this:

```
C:\orant\bin>exp71
```

```
Export: Release 7.1.3.3.3 - Production on Tue Sep 05 17:21:28 1995
```

```
Copyright  Oracle Corporation 1979, 1994.  All rights reserved.
```

```
Username: jgreene  
Password:
```

```
Connected to: Oracle7 Workgroup Server Release 7.1.3.3.3 - Production Release
PL/SQL Release 2.1.3.2.1 - Production
Enter array fetch buffer size: 4096 >
```

```
Export file: EXPDAT.DMP > jgreene.dmp
```

```
(1)E(ntire database), (2)U(sers), or (3)T(ables): (2)U > u
```

```
Export grants (yes/no): yes >
```

```
Export table data (yes/no): yes >
```

```
Compress extents (yes/no): yes >
```

```
About to export specified users ...
User to be exported: (RETURN to quit) > JGREENE
```

```
About to export JGREENE's objects ...
. exporting snapshots
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export JGREENE's tables ...
. exporting table          GOLF_SCORES      120 rows exported
. exporting table          TEST1            110 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
User to be exported: (RETURN to quit) >
```

```
. exporting referential integrity constraints
. exporting triggers
. exporting posttables actions
Export terminated successfully without warnings.
```

Then, to import the data, it would go something like the following:

```
C:\orant\bin>imp71
```

```
Import: Release 7.1.3.3.3 - Production on Tue Sep 05 17:22:45 1995
```

```
Copyright Oracle Corporation 1979, 1994. All rights reserved.
```

```
Username: jgreene
Password:
```

```
Connected to: Oracle7 Workgroup Server Release 7.1.3.3.3 - Production Release
PL/SQL Release 2.1.3.2.1 - Production
```

```
Import file: EXPDAT.DMP > jgreene.dmp
```

```
Enter insert buffer size (minimum is 4096) 30720>
```

```
Export file created by EXPORT:V07.01.03
```

```
List contents of import file only (yes/no): no >
```

```
Ignore create error due to object existence (yes/no): yes >
```

```
Import grants (yes/no): yes >
```

```
Import table data (yes/no): yes >
```

```
Import entire export file (yes/no): yes >
```

```
. importing JGREENE's objects into JGREENE
```

```
. . importing table "GOLF_SCORES"
```

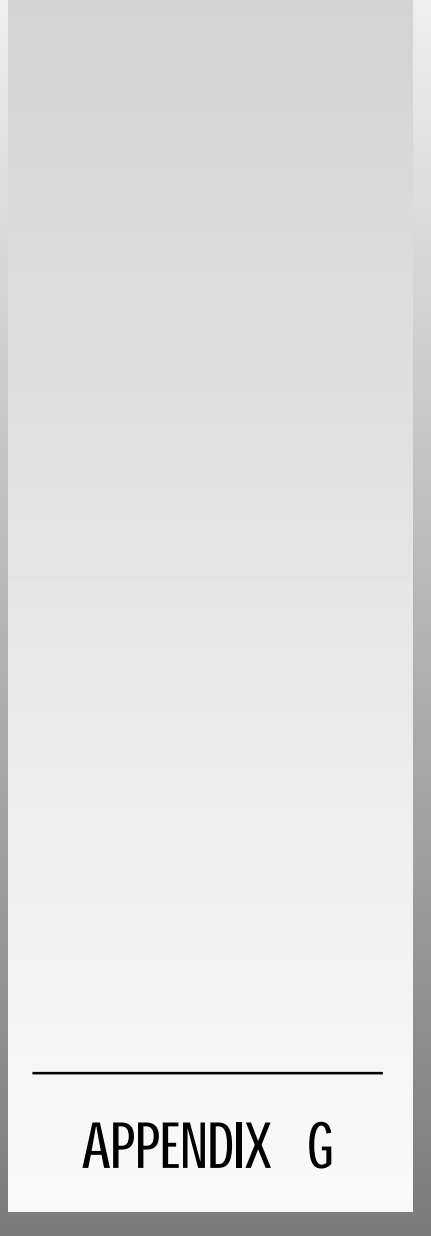
120 rows imported

```
. . importing table "TEST1"
```

110 rows imported

```
Import terminated successfully without warnings.
```

```
C:\orant\bin>
```

Where to Get More Information

Sometimes it is not as important to know something off the top of your head as it is to know where you can find that information. This appendix lists a few useful sources to get you started.

CONVENTIONAL ADDRESSES

Oracle Corporation (General Information)

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

USA

Phone: (415) 506-7000

Fax: (415) 506-7200

Oracle Asia/Pacific Rim/India Headquarters

Phone: +65.220.5488

Fax: +65.227.4098

Oracle Europe/Middle East/Africa

Phone: +31.34.069.4211

Fax: +31.34.066.5603

Oracle Technical Support Number

(415) 506-1500 (have your CSI number handy)

Oracle Magazine

500 Oracle Parkway

Box 659510

Redwood Shores, CA 94065

FTP SITES

`ftp.informatik.uni-oldenburg.de/pub/oracle`

`ftp.ml.csiro.au/softwarejstander/oracle_utilities`

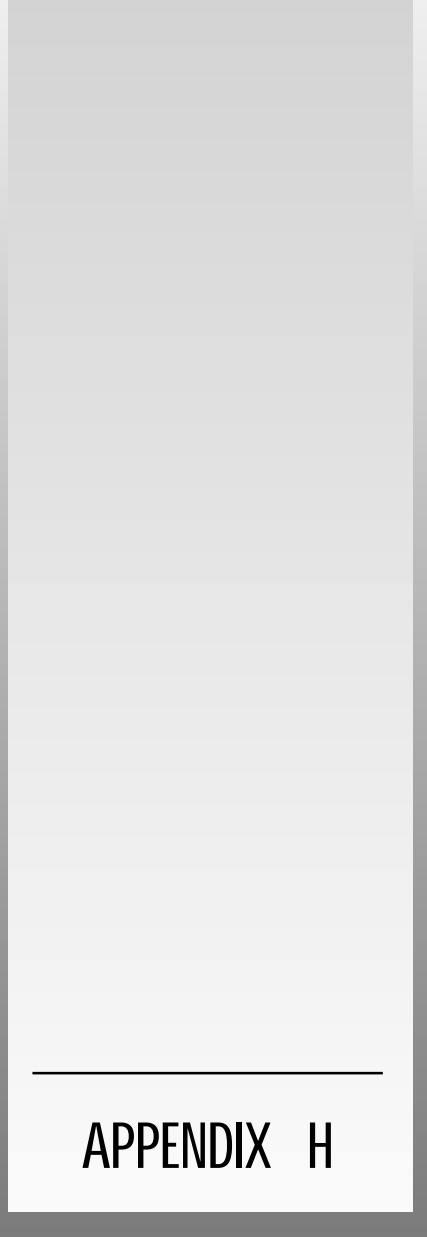
WORLD WIDE WEB PAGES

`http://www.oracle.com/` [Oracle corporation itself]

`http://www.ioug.org/` [International Oracle Users Group]

NEWSGROUPS

`comp.database.oracle`



Sample System Configuration Analyses

Customer XYZ
 HP Oracle Installation Analysis
DRAFT 1/24/93

1. Overview

This document seeks to capture the design rationale used when installing and configuring the Oracle Relational Data Base Management System on the HP 9000 series 800 server at customer XYZ.

2. Overall System Conceptual Design

Several applications are being developed or converted to run in a client-server environment. The HP 800 computer will act as the database server in this environment. In addition, several off-the-shelf applications may be purchased and loaded on this computer. Some of these applications may use the client-server model and others may require the users to log in to the HP 800 as a terminal and execute the applications on the HP 800 itself.

3. Oracle Installation Conceptual Design

Instances will be arranged based on development, test, and production needs.

4. Initial Configuration Recommendation

Install only a development instance. Ensure that the methodology allows expansion to future instances.

5. Significant Installation Notes

a. Oracle 7.0.13 requires HP-UX 9.0.

b. Dependencies:

Oracle*Terminal	Oracle 7 server Orakit
SQL*Plus	Oracle 7 server Help tables
Pro*C	Oracle 7 server HP-C compiler version 8.74
SQL*Net SPX	Oracle 7 server Netware 9000
SQL*Net TCP	Oracle 7 server HP LAN/9000 link and OpenDNA 4.5.7
PL/SQL	Oracle 7 server

c. Space Requirements

Product	Disk (MB)	DB (MB)	Memory (kB)
Oracle server	35.28	12.5	4,456
Dist Opt	0.08	0	1,828
Proc Op	0.27	0	1,476
Toolkit II	14.24	0	1,363
Common Lib	9.15	0	1,340
Mig. Util	2.01	0	0
PL/SQL	0.08	0	807
SQL*Net TCP	0.73	0	0
Terminal	3.6	0	920
SQL*Plus	4.47	3.0	1,617
Pro*C	3.49	0..	2,437
Totals	84	12.5	N/A*

*Not applicable because many of the packages, such as SQL*Plus, may not be in use at all times.

- d. SQL*Plus 3.1 or later is needed for Oracle 7.
 - e. After loading, but before installing, in .login file, set LDOPTS=" -a archive";export LDOPTS. Then log in and log out.
 - f. Purchase includes Oracle RDBMS version 7.0 for HP 98xx/HP-UX 9.0, Model H50 for 32 concurrent users full-use license.
 - Procedural database option version 7.0
 - SQL*Plus version 3.1
 - SQL*Net version 1.2
 - SQL*Net SPX/IPX version 1.2
 - PRO*C version 1.5
 - g. The CSI # is 123456 (Good through 12/27/94).
 - Oracle support # (415) 506-1500
 - PO # 99999
6. Initial Oracle Instance Configuration Options
- Disk drive configurations:
- | | |
|--------------------------|-----------------------------------|
| /usr/oracle | Oracle base application directory |
| /usr/oracle/product | Oracle home directory |
| /usr/oracle/product/7013 | Install point for this release |

/d1/oracle/devel	Development instance directory
/d2/oracle/devel	Development instance directory
/d3/oracle/devel/redo	Storage for online redo logs
/d3/oracle/devel/archive	Storage for archive logs
/d4	Future use
/d5	Future use

7. Preparation Tasks

- a. HP admin create dba group.
- b. HP admin create data base administrator logins within dba group.
- c. HP admin create oracle owner account in dba group.
- d. HP admin create oracle user logins. Note, some thought needs to go in to how the regular user groups will be arranged. There is no need to worry about users who will access only via client-server. This applies to developers and others who will log in to the server.
- e. HP admin create files in figure 2-3 on page 2-9.
- f. HP admin alter the UNIX kernel parameters as follows:

SHMMAX	0x4000000	{4 million}
SHMMNI	100	
SHMSEG	12	
SEMMNS	128	
SEMMNI	10	
- g. Root user create all mount point directories (/d1, /d2, etc).
- h. Root user create the directories listed in 6a above with owner oracle and mode 755.
- i. User oracle set ORACLE_BASE environmental variable.
- j. Create the following subdirectories in ORACLE_BASE:
 - product
 - admin
 - data
 - local
 - TAR
- k. Create version 7013 subdirectory under ORACLE_BASE/product.
- l. Set ORACLE_HOME environmental variable to ORACLE_BASE/product/7013.

- 
- adduser
 - alert
 - bench
 - config.sql
 - dbch_bat
 - dbcheck
 - dbcheck.2
 - grants.sql
 - login.sql
 - oracle.cro
 - orastat
 - orastat.2
 - pccomp
 - etc.

APPENDIX I

The Disk Contents

This appendix provides an introduction to the contents of the enclosed disk and comments as to what you have to modify to customize these scripts to fit into your local environment. The following are the scripts contained on the disk:

adduser (UNIX shell script, not needed in Personal Oracle7)

This is a convenient script that enables you to add a batch of users to the system. It prompts you for the user ID, password, and role(s) that will be assigned. It then fills in the default tablespace and the temporary tablespace based on values that you store in the script. It creates a SQL script and then executes this script for you. The parameters that you have to customize in the file to fit your needs are the following:

```
default_ts=users
temp_ts=temp
```

alert (UNIX shell script)

This script makes a copy of the current alert file for an instance that has today's date as part of the title. It then zeros out the alert log file to make it ready to receive new data. The parameters that you have to customize to fit your needs are the following:

```
DBA=/oracle/admi      [Directory to store the alert log files]
instance=test         [sid of your instance]
```

bench (UNIX shell script)

This script enables you to run a SQL script that has been prepared with a statement to activate the SQL trace utility, figure out which trace file is associated with the job that you have run, and execute the explain plan utilities. The parameter that you have to customize to fit your needs is the following:

```
trace_dir=/info/oracle/admin/test/udump
```

config.sql (SQL script)

This script runs the configuration report discussed in Part 6 of this book. There are no parameters that need to be customized to fit your local needs.

dbch_bat (UNIX shell script)

This script implements the database checking utilities from a batch file format. A file is required that stores the Oracle system password (protected with 400 privileges). You need to input parameters—such as the SID, ORACLE HOME, where the scripts are located, which report to run (*option* = u for utilization, c for configuration, t for tuning, and s for security). This report automatically prints the output or sends e-mail to the users specified in the parameters list. The following parameters need to be customized for the report that you want to run:

```
password_file=/info/oracle/admin/bin/sxsystem
ORACLE_SID=test
```



```
ORACLE_HOME=/disk1/oracle/product/7.0.15
script_dir=/info/oracle/admin/bin
output_dir=/info/oracle/rptlogs
option=u
printit=n
user=system
subject_line="Utilization_report_for_test"
mail1=jogreen
mail2=NONE
output_file="util.test"
```

dbcheck (UNIX shell script)

This script is the interactive version of the control script, which enables you to run a specified database monitoring report on a given instance. You can send the results directly to a printer, if desired. You need to adjust the following parameters to fit your environment:

```
ORACLE_SID=test1                # Default only, user can override
script_dir=/info/oracle/admin/bin # Where are these scripts stored
output_dir=/info/oracle/rptlogs   # Where the reports are stored
```

dbcheck.2 (UNIX shell script)

This is the shell script that does most of the actual work of running the appropriate database monitoring report, printing the output, or mailing the results to the user. It is called by both the dbcheck and dbch_bat scripts. There are no parameters that need to be adjusted to fit your local environment.

grants.sql (SQL script)

This is a sample SQL script that grants the appropriate set of privileges for one of the examples used in the book. Use it as an idea for when you build your own grant scripts.

login.sql (SQL*Plus initialization script)

This is an initialization file that can be used for SQL*Plus. It is an example of how I like to set up my SQL*Plus sessions. You can customize it to suit your own tastes.

oracle.cro (UNIX table)

This is a sample UNIX cron tab file that I use to run most of my automated reports and database maintenance jobs. Comments at the top make this file readable and therefore easily modified. Feel free to use it as a template for your own cron tab files.

orastat (UNIX shell script)

This is a shell script that contains one line (it calls the orastat.2 script). I use this trick to enable me to send the output results through the UNIX `more` command, which displays the output one screen at a time.

orastat.2 (UNIX shell script)

This is a shell script that uses the Unix process status (`ps`) command and `grep` utilities to display a list of the UNIX processes that are running. You need to insert your Oracle instance names in these `grep` statements. However, if you put this and the `orastat` script in a directory that is in your `PATH`, you can type `orastat` and obtain a list of the running Oracle background processes from anywhere in your system.

pccomp (UNIX shell script)

This is a convenient interface to the Pro*C Compiler. All it does is save me from having to remember that long directory path that is needed to specify where the make file is located.

prob_chk.txt (ASCII text file)

This is the problem-solving checklist described in Chapter 38. You can edit this to suit your needs and preferences in your favorite word processor.

process.sql (SQL script)

This is an interesting little SQL query that displays the active Oracle processes and their associated user IDs.

sec_ck.sql (SQL script)

This is the security report described in Chapter 30 on monitoring your Oracle database.

sec_role.txt (ASCII Text File)

This script is a little checklist discussed in Chapters 21 and 26 related to building up a security scheme and roles.

sizer.sql (SQL script)

This is a template for a script that determines the average size of a row in the table. You need to fill in all the column names and the table name. This is needed to determine accurately the amount of space that is actually being used.

starter (UNIX shell script)

This is a UNIX shell script that starts or stops your Oracle instance, based on the value passed into it (`start` or `stop`). It can be a handy tool when you don't want to worry about remembering `SQL*DBA` commands or menus. The parameters that you need to customize for your local needs are the following:

```
ORACLE_HOME=/disk1/oracle/product/7.0.15
SYSTEM_BIN=/disk1/oracle/admin/bin
ORACLE_SID=test
SCRIPT_NAME=starter
```

t_cklist.txt (ASCII text file)

This script is a text-based tuning checklist.

tab_size.xls (Microsoft Excel 4 spreadsheet)

This script is a spreadsheet that stores the algorithms needed to predict accurately the size of a set of tables and indexes, based on the current number of rows, the expected number of rows, and the current size per row.

tune_ck.sql (SQL script)

This SQL script produces the tuning report described in Chapter 30.

util_ck.sql (SQL script)

This SQL script produces the utilization report described in Chapter 30.



SYMBOLS

4GL (fourth-generation language) tools, 5

A

accept command, 369

accept variable

format prompt text command, 708

accuracy of data, 37

adding

data files (troubleshooting tablespaces), 553-555

new users (user account maintenance), 399-402

privileges (user account maintenance), 402-406

adduser script, 732

Admin directory, 126

adminoption

(granting privileges), 208

admin tree, 286

alert logfiles, 702

alert script, 732

ALL_ROWS hint, 622

ALTER ANY CLUSTER

privilege, 181

ALTER ANY INDEX

privilege, 181

ALTER ANY PROCEDURE privilege, 182

Index

ALTER ANY ROLE

privilege, 182

ALTER ANY SEQUENCE

privilege, 183

ALTER ANY SNAPSHOT

privilege, 183

ALTER ANY TABLE

privilege, 183

alter database archivelog/
noarchivelog command,
699

ALTER DATABASE

privilege, 186

alter index command,
698-699

ALTER privilege, 192

sequences, 198

tables, 197

ALTER PROFILE

privilege, 186

alter redo log

command, 699

ALTER RESOURCE COST

privilege, 186

alter rollback segment

command, 699

ALTER ROLLBACK

SEGMENT privilege, 186

ALTER SESSION

privilege, 178

ALTER SYSTEM

privilege, 186

alter system switch logfile
command, 699

alter table command,
440-441, 698

alter tablespace add
datafile command, 696

alter tablespace
command, 236, 554, 698

ALTER TABLESPACE

privilege, 187

ALTER USER

privilege, 187

ANALYZE ANY

privilege, 181

analyze table

command, 414

AND_EQUAL hint, 623

“any” privileges, 177,
180-185, 189-190

application design,

optimizing queries, 616

application developers, 28

application development
tools, 61-63

application turnover
(software installation),
297

applications

interfaces to non-Oracle

applications, 6

memory allocations,
119-120

processing requirements
(tuning databases),
501-502

troubleshooting, 564-573
adjusting applications,
571

classifying problem
causes, 564-566

expanding Oracle
resources, 570

log files, 566-569

operating system
conflicts, 569-570

technical support,
571-573

tuning (database speed),
580-581

applying patches, 302**ARCH (archivers), 148****architectures, 57-61**

client-server architectures,
59, 631, 674-678

advantages, 677-678

client configuration, 274
compared to distributed
databases, 680

database administra-
tion, 679-680

designing, 273

evaluating environ-
ments, 271-272

examples, 678-679

implementing

Oracle, 71

installation, 272

networks, 676

server configuration,
273-274

see also host-based
architectures, 271

data warehouses, 60

databases, 59-60

host-based, 58-59, 70, 268
advantages / disadvan-
tages, 269

evaluating

environment, 270

implementing

Oracle, 70

installations, trouble-
shooting, 270

software, configuring,
269

VAX system

configurations, 271

implementing, 69-73

OFA (Optimal Flexible
Architecture), 125-127,
285-287

processing architectures,
60-61

archive log files, 105, 124, 131-132, 702

- advantages/disadvantages, 228-229
- archivers, 148
- copying redo log files, 129-131
- database administration schemes, 346
- selecting backup schemes, 240-241

archive log status, modifying, 699**archiver process, 290****archivers, 141, 148****asynchronous database writers, 660****at utility (UNIX), 374****AUDIT ANY privilege, 181****audit command, 484-488****audit files, 125****AUDIT SYSTEM****privilege, 187****auditing databases, 448, 454-455, 480-494**

- events, 480-491
- monitoring audit trails, 492-494
- object auditing options, 486-488
- performance, 491
- security, 482-483, 491-492
- shortcuts, 488-489
- starting, 481-482
- statement auditing options, 484-486
- system privileges, 484
- views, 482, 490-491

authenticating**passwords, 397****automated backup schemes, 245-247****automated job submission utilities, database administration schemes, 373-377****automatic locks, 652****automating DBA tasks, scripts, 365-367****availability hours (computer down-time), database administration schemes, 335-336**

B**background processes, 99-100, 138-151**

- data writing processes, 138-140, 144-146
- dedicated servers, 149
- lock writers, 148-149
- log writing processes, 138-141, 147-148
 - archivers, 148*
 - recoverers, 148*
- monitoring processes, 138, 141-142
 - process monitors, 144*
 - system monitors, 142-143*
- multi-threaded servers, 149-150
- parallel query processes, 150-151
- SQL*Net listeners, 150
- user service processes, 138-139

BACKUP ANY TABLE privilege, 184**Backup Manager, 64, 83-84****backup schemes**

- automated backup schemes, 245-247

- rotating backup schemes, 242-245

- selecting, 239-242

backup/recovery (DBA job description), 21, 27, 31-34, 77**backups, 702, 224-248**

- advantages, 224-225
- archive log files, 228-229
- automated backup schemes, 245-247
- cold backups, 226, 229-236
 - files needed, 232*
 - recovery of data, 233-236*
- consistency of data, 226
- Export utility, 227, 238-239
- installing software, 298-299
- object-by-object backups, 226-227, 238-239
- operating system file backups, 226, 231-232
- partial backups, recovery of data, 234-235
- Personal Oracle7, 232-233
- recovery of data, 225
- rotating backup schemes, 242-245
- selecting backup schemes, 239-242
- testing backups, 235-236
- upgrading software, 324
- warm backups, 226, 236-238
 - ending, 237*
 - starting, 236*

badfile optional parameter (SQL*Loader), 717**balanced disk loading, tuning databases, 502**

BECOME USER

- privilege, 187
- bench script, 732
- books (DBA continuing education), 634
- bridges, 676
- browser tools, 62-63
- budget presentations (planning database expansion), 523-527, 581-583
- buffer cache hit ratio (tuning databases), 144, 509
- bugs
 - fixing, upgrading software, 315
 - Installer, 301
- business needs
 - installation procedures, 279-281
 - collecting data*, 280
 - specific details of users*, 279
 - user requirements*, 280-281
- by clause (auditing databases), 489

C**caching tables in**

- memory, 114

calculating

- data space (installation procedures), 288
- disk space requirements, 410-416
- index sizes, 429-430
- system requirements for database expansion, 522-523

- calling SQL*Plus, 706

Carnegie Mellon

- University's Software Engineering Institute, 450

- CASE (computer-aided software engineering), 5, 68

- cat command, 379

- CATEXP.SQL script, 721

- CDROMs (installing software), 301-303

- char data type, 161

checklists

- troubleshooting Oracle, 586-589
- tuning databases, 513-514

checkpoint process, 290**classifying problem**

- causes, troubleshooting

- instance/application

- crashes, 536-539, 564-566

- client-server architectures, 59, 271, 631,

- 674-678

- advantages, 677-678

- compared to distributed databases, 680

- configuration

- client configuration*, 274

- server configuration*, 273-274

- database administration, 679-680

- designing considerations, 273

- evaluating environments, 271-272

- examples, 678-679

- implementing Oracle, 71

- installation problems, 272

- networks, 676

- see also* host-based architectures

- clusters (of tables), 158, 168

- CODASYL (data language standard), 47

- cold backups, 226, 229-236

- files needed, 232

- recovery of data, 233-236

- selecting backup

- schemes, 240

- column command,

- 368, 707-708

- column privileges, 198

- columns (tables)

- formatting, 707-708

- column command*, 368

- SQL*Plus*

- commands*, 89

- naming, 160

- command auditing options (auditing databases), 484-486

- command line mode

- (Import utility), 720

- command line mode

- (SQL*DBA), 712

- command-line interface,

- see* SQL*Plus, 88

commands

- accept, 369

- accept variable format

- prompt text, 708

- alter database archivelog/noarchivelog, 699

- alter index, 698-699

- alter redo log, 699

- alter rollback

- segment, 699

- alter system switch

- logfile, 699

- alter table, 440-441, 698
- alter tablespace, 236, 554, 698
- alter tablespace add datafile, 696
- analyze table, 414
- audit, 484, 486, 488
- cat, 379
- column, 707, 708, 368
- create index, 163, 432-433, 603-604, 697-698
- create public rollback segment, 644
- create role, 210, 401
- create table, 159-160, 431-432, 696-697
 - creating implied indexes, 603*
- create tablespace, 696
- create user, 399, 400
- create view, 165
- crontab, 375
- drop index, 441, 700
- drop role, 211
- drop rollback segment, 700
- drop table, 441, 552, 699
- drop tablespace, 700
- drop user, 406-407
- edit, 89
- get, 89
- grant, 206-209, 401
- lp (line printer), 379
- mailx, 379
- noaudit, 490
- prompt, 708, 368
- ps (process status), 378
- read, 371
- revoke, 207, 402
- save, 89
- script files, 93
- set pagesize, 706-707, 89
- set pause, 707, 89

- set role, 211
- show sga, 117
- spool, 89, 368, 707-708
- sqlplus, 706
- COMMENT ANYTABLE privilege, 184**
- common problems, troubleshooting, 543-544**
- compressing extents, 555-559**
- computer-aided software engineering (CASE) tools, 5, 68**
- computer system support, 18**
- computing power (tuning databases), 499-500**
- concentrators, 676**
- conceptual tools, 76-78**
- concurrency (locks), 650-651**
- config.ora file, 133**
- config.sqlscript, 732**
- configuration report (monitoring databases), 454, 474-477**
- configurations**
 - client-server architectures
 - client configuration, 274*
 - server configuration, 273-274*
 - database administration schemes, 340-341, 346-348
 - demonstrations from vendors, 275
 - host-based architectures, 269
 - VAX system, 271*
 - installing software, 298
 - system configuration analyses, 728-730

- conflicts with schedules (database administration schemes), 357-359**
- CONNECT (Oracle 6 privilege set), 177**
- consistency**
 - data backups, 226
 - locks, 650-651
- continuing education (DBAs), 22, 630-638, 726**
 - books, 634
 - Internet, 635-637
 - Oracle user groups, 637-638
 - test instances, 635
 - vendor training programs, 632-634
- control files, 105, 124, 133**
 - database administration schemes, 344-345
 - example control file (SQL*Loader), 717
- cooked data files, 128**
- corruption of data (database security), 35**
- cost-based optimizers, 614, 617-618**
- CPU utilization, tuning databases, 504-505**
- CREATE ANY CLUSTER privilege, 181**
- CREATE ANY INDEX privilege, 181**
- CREATE ANY PROCEDURE privilege, 182**
- CREATE ANY SEQUENCE privilege, 183**
- CREATE ANY SNAPSHOT privilege, 183**
- CREATE ANY SYNONYM privilege, 183**

CREATE ANY TABLE
 privilege, 183

CREATE ANY TRIGGER
 privilege, 184

CREATE ANY VIEW
 privilege, 185

CREATE CLUSTER
 privilege, 179

CREATE DATABASE
 LINK privilege, 179

create index command,
 163, 432-433, 603-604,
 697-698

CREATE PROCEDURE
 privilege, 179

CREATE PROFILE privilege,
 186

CREATE PUBLIC
 DATABASE LINK privilege,
 186

create public rollback
segment command, 644

CREATE PUBLIC SYN-
ONYM privilege, 179

CREATE ROLE
 command, 210, 401
 privilege, 186

CREATE ROLLBACK
SEGMENT privilege, 186

CREATE SEQUENCE
 privilege, 179

CREATE SESSION privilege,
 178

CREATE SNAPSHOT
 privilege, 180

CREATE SYNONYM
 privilege, 180

CREATE TABLE
 command, 159-160,
 431-432, 696-697
creating implied
indexes, 603
 privilege, 180

CREATE TABLESPACE
 command, 696
 privilege, 187

CREATE TRIGGER
 privilege, 180

create user command,
 399-400

CREATE VIEW
 command, 165
 privilege, 180

creating
 data files, 696
 databases
Installer, 304-305
manually, 307-309
 instances, software
 installation, 298
 rollback segments, 643,
 647-648
 see also designing

cron utility (UNIX),
 374-377

crontab command, 375

CSI (customer service ID)
 number, 702

cursors, shared pool (SGA
memory), 116

D

daily maintenance sched-
ules (database adminis-
tration schemes), 359-360

daily routine overview
(DBAs), 384-391
 monitoring objects,
 386-387
 scheduled events, 386
 troubleshooting, 388-390
 user support, 387-388

data
 accuracy, 37
 corruption (database
 security), 35

independence, 597

integrity (rollback seg-
 ments), 642-643

storing, 99, 104-106,
 154-170
clusters, 158, 168
databases, 154
history, 44-46
indexes, 158, 162-164
rollback segments,
 644-645
sequences, 158, 169
stored procedures, 158,
 167-168
synonyms, 158, 166-167
tables, 158, 159-162
tablespaces, 155-157
views, 158, 164-166

transferring
 (SQL*Loader), 63, 78,
 91-92, 716-717
badfile optional param-
eter, 717
example control file, 717
fixed column formats,
compared to delimited
formats, 716
load tables, 716
log file optional param-
eter, 717
population scripts, 716

Data Definition Language (DDL), 702

saving scripts, 560

data dictionaries, 702

data dictionary cache, 116
 miss rate (tuning data-
 bases), 507-508

Data directory, 126

data files, 104-105, 124, 127-129

adding (troubleshooting
 tablespaces), 553-555
 creating, 696

- database administration
 - schemes, 333, 341-344
- purging old data files, 559
- recovery of data, 234
- tablespace maintenance, 410-422
 - calculating disk space requirements, 410-416*
 - monitoring tablespaces, 416-418*
 - troubleshooting, 418-421*
- writing to, 101
- Data Manipulation Language (DML), 702**
- data types, 161**
- data updates (DBA job description), 32**
- data warehouses, 77**
 - computer architectures, 60
- data writing processes, 138-140, 144-146**
 - asynchronous database writers, 660
- database administration schemes, 330-338**
 - archive log files, 346
 - client-server architectures, 679-680
 - configuring databases, 340-341, 346-348
 - control files, 344-345
 - data files, 333, 341-344
 - definition, 330-333
 - expanding databases, 348
 - maintenance schedules, 352-362
 - daily schedules, 359-360*
 - long-term schedules, 360-361*
 - scheduling conflicts, 357-359*
 - user processing schedules, 355-357*
 - meeting user requirements, 335-336
 - naming conventions, 332
 - normalized tables, 333-334
 - OFA (Optimal Flexible Architecture), 340-341
 - planning, 332-333, 336-338
 - redo log files, 345-346
 - referential integrity, 334
 - scripts, 364-380
 - automated job submission utilities, 373-377*
 - automating DBA tasks, 365-367*
 - monitoring script results, 378-379*
 - PL/SQL scripts, 369-370*
 - shell scripts, 370-373*
 - SQL scripts, 367-369*
 - testing scripts, 377-378*
 - security, 334
 - sorting data, 334
 - troubleshooting, 332
 - warm backups, 343
- database administrators (DBAs)**
 - conceptual tools, 76-78
 - continuing education, 726, 630-638
 - books, 634*
 - Internet, 635-637*
 - Oracle user groups, 637-638*
 - test instances, 635*
 - vendor training programs, 632-634*
 - daily routine overview, 384-391
 - monitoring objects, 386-387*
 - scheduled events, 386*
 - troubleshooting, 388-390*
 - user support, 387-388*
- history, 7-8, 14-15
- job description, 8-10, 14-42, 683-684
 - backup/recovery, 21, 27, 31, 33-34*
- computer system
 - support, 18
- continuing education, 22
- data accuracy, 37
- database performance, 35-36
- developer as database administrator, 23-28
- developer support, 17-18
- expertise level
 - expected, 17
- full-time database administrators, 18-23
- logical data
 - structures, 37
- monitoring database health, 21, 27, 32
- part-time database administrators, 28-32
- percentage of time
 - available, 17
- planning for future, 21
- reviewing new products, 21-22, 28
- security of database, 33-35
- size of database
 - system, 17
- specialization requirements, 18
- support staff relationships, 37-39
- troubleshooting, 21, 27, 32
- updating data, 32

- user administration*, 21, 27, 32
- user support*, 21, 28, 39-42
- tips/suggestions summary, 684
- database buffer cache**, 113-114
 - data writing processes, 144-146
- Database Expander**, 64, 85-86
- database maintenance privileges**, 177, 185-187
- database management systems, history**, 45-48
- Database Manager**, 64, 83
- databases**, 154, 702
 - adding new users, 399-402
 - auditing, 448, 454-455, 480-494
 - events*, 480-481, 483-491
 - monitoring audit trails*, 492-494
 - object auditing options*, 486-488
 - performance*, 491
 - security*, 482-483, 491-492
 - shortcuts*, 488-489
 - starting audits*, 481-482
 - statement auditing options*, 484-486
 - system privileges*, 484
 - views*, 482, 490-491
 - backing up
 - advantages*, 224-225
 - cold backups*, 229-236
 - consistency of data*, 226
 - computer architectures, 59-60
 - configuring, system configuration analyses, 728-730
 - creating
 - Installer*, 304-305
 - manually*, 307-309
 - definition, 56-57, 99
 - distributed databases
 - compared to parallel servers*, 658-659
 - installing Oracle*, 279
 - parallel processing*, 657
 - recoverers*, 148
 - expanding
 - database administration schemes*, 348
 - troubleshooting database speed*, 581-583
 - future developments, 52
 - health of databases, 448-451
 - installation
 - business needs considerations*, 279-281
 - calculating data space*, 288
 - environments, evaluating*, 264
 - layout recommended by Oracle*, 285-287
 - memory planning*, 289
 - process planning*, 289-290
 - products, evaluating*, 266-267
 - purchasing components*, 262-267
 - troubleshooting*, 265-266
 - user requirements*, 280-281
 - life cycles, 253
 - installation*, 253, 255-257
 - operations*, 253
 - product selection*, 253, 254
 - upgrade*, 253, 257-260
 - market share, 51-52
 - monitoring, 102, 451-454, 458-460
 - configuration report*, 474-477
 - database tuning reports*, 465-470
 - future planning*, 518-527
 - scheduling reports*, 459-460
 - scripts/reports*, 460-477
 - security reports*, 470-474
 - third-party tools*, 477-478
 - tuning report*, 497-498
 - utilization report*, 462-465
 - mounting, 154
 - multimedia databases, 52
 - naming conventions, 604-605
 - normalizing, 702, 425, 597-599
 - object-oriented databases, 52
 - performance, 35-36, 449-451
 - physical database structure, 426-431
 - compared to logical*, 426
 - privileges, 194
 - reliability, 450

- searching, 162-164
- security, 34, 451
 - auditing databases, 482-483, 491-492
 - data corruption, 35
 - database administration schemes, 330-334
 - DBA job description, 33-35
 - disaster recovery, 35
 - granting privileges, 206-209, 212
 - health of databases, 451
 - object privileges, 176, 192-204
 - passwords, 34
 - reports, 453, 470-474
 - roles (granting privileges), 209-212
 - scripts, password files, 378
 - stored procedures, 668
 - system privileges, 172-190
 - user account maintenance, 394-396
- software storage, 664-666, 671
 - functions, 664
 - object-oriented programming, 666
 - packages, 665-666, 670-671
 - stored procedures, 664, 668-670
 - triggers, 665-667
- speed
 - application tuning, 580-581
 - efficiency measuring, 576-578
 - troubleshooting, 576-584
 - tuning databases, 579-580
 - user expectations, 578-579
- troubleshooting, 532-545
 - adding data files, 553-555
 - allocating space from other tablespaces, 559
 - checklist, 586-589
 - classifying problem causes, 536-539
 - common problems, 543-544
 - compressing extents, 555-559
 - database speed, 576-584
 - de-fragmentation, 555-559
 - dropping tables, 551-552
 - log files, 536-538
 - prioritizing problems, 533-536
 - purging old data files, 559
 - saving DDL scripts, 560
 - service level agreements, 544-545
 - tablespaces, 548-561
 - technical support, 539-543
 - unknown problems, 541-542
 - utilization report (tablespace extents), 548-551
- tuning, 448, 455, 496-515
 - application processing requirements, 501-502
 - balanced disk loading, 502
 - buffer cache hit ratio, 509
 - checklist, 513-514
 - computing power, 499-500
 - CPU utilization, 504-505
 - data dictionary cache miss rate, 507-508
 - disk space, 500-501
 - disk striping, 503
 - free list, 513
 - hierarchical storage, 503
 - I/O operations, 505, 509-510
 - latch contention, 511
 - library cache miss rate, 507
 - memory utilization, 500, 502, 505
 - multi-threaded server processes, 511
 - multi-threaded server session memory, 508-509
 - operating system indicators, 504-506
 - redo log buffer space contention, 511-512
 - reports, 452-453, 465-470
 - resource contention, 506-513
 - rollback segments, 510
 - sort memory contention, 512-513
 - upgrading, 315, 325
- date data type, 161**

DBA (Oracle 6 privilege set), 177**DBAs**

- conceptual tools, 76-78
- continuing education, 630-638, 726
 - books, 634
 - Internet, 635-637
 - Oracle user groups, 637-638
 - test instances, 635
 - vendor training programs, 632-634
- daily routine overview, 384-391
 - monitoring objects, 386-387
 - scheduled events, 386
 - troubleshooting, 388-390
 - user support, 387-388
- history, 14-15
- job description, 14-42, 683-684
 - backup / recovery, 21, 27, 31, 33-34
 - computer system support, 18
 - continuing education, 22
 - data accuracy, 37
 - database performance, 35-36
 - developer as database administrator, 23-28
 - developer support, 17-18
 - expertise level expected, 17
 - full-time database administrators, 18-23
 - logical data structures, 37

- monitoring database health, 21, 27, 32
- part-time database administrators, 28-32
- percentage of time available, 17
- planning for future, 21
- reviewing new products, 21-22, 28
- security of database, 33-35
- size of database system, 17
- specialization requirements, 18
- support staff relationships, 37-39
- troubleshooting, 21, 27, 32
- updating data, 32
- user administration, 21, 27, 32
- user support, 21, 28, 39-42

- tips/suggestions summary, 684

dbch_bat script, 732-733**dbcheck script, 733****dbcheck.2 script, 733****DBWR (database writer), 144-146****DDL (Data Definition Language), 702**

- saving scripts, 560

de-fragmenting

- tables, 438-439

- tablespaces, 420, 555-559

dedicated servers, 139, 149**default roles, 210, 211****default storage parameters (tablespaces), 156-157****definition of Oracle, 682-683****DELETE ANY TABLE privilege, 184****DELETE privilege, 192**

- tables, 197

- views, 197

deleting

- indexes, 441, 700
- roles, 211
- rollback segments, 645, 700
- tables, 441, 552, 699
- tablespaces, 700
- user accounts, 406-407

delimited formats

- (SQL*Loader), compared to fixed column formats, 716

designing

- applications, optimizing queries, 616
- client-server architectures, 273
- indexes/tables, 424-434
- installation procedures, 290-292
 - reviewers, 292
- tables for decision support, 600-601
- see also creating

Developer 2000, 62**developer as database administrator (job description), 23-28****developer privileges, 177-180****Development Group**

- Database Administrators (job description), 10

directories

- file storage, 287
- naming conventions, 287

ORACLE HOME, 285
 software storage,
 285-287, 314
 admin tree, 286
 ORACLE HOME
 directory, 285
 sub tree, 286

dirty buffers (data writing processes), 144-146

disabling user accounts temporarily, 407

disaster recovery, 702, 35

disk drives
 data storage, 104-106
 naming conventions, 287
 storing files, 287
 user requirements for
 installation, 281

disks
 mirroring, 228, 247
 space
 calculating, 288,
 410-416
 fragmentation, 418,
 419-421
 index sizes, 429-430
 tuning databases,
 500-501
 striping (tuning data-
 bases), 503

distributed databases
 comparisons
 to client-server architec-
 tures, 680
 to parallel servers,
 658-659
 installing Oracle, 279
 parallel processing, 657
 recoverers, 148

DML (Data Manipulation Language), 702

DROP ANY CLUSTER privilege, 181

DROP ANY INDEX privilege, 181

DROP ANY PROCEDURE privilege, 182

DROP ANY ROLE privilege, 183

DROP ANY SEQUENCE privilege, 183

DROP ANY SNAPSHOT privilege, 183

DROP ANY SYNONYM privilege, 183

DROP ANY TABLE privilege, 184

DROP ANY TRIGGER privilege, 184

DROP ANY VIEW privilege, 185

drop index command, 441, 700

DROP PROFILE privilege, 186

DROP PUBLIC DATABASE LINK privilege, 186

DROP PUBLIC SYNONYM privilege, 179

drop role command, 211

DROP ROLLBACK SEGMENT
 command, 700
 privilege, 186

drop table command, 441, 552, 699

drop tablespace command, 700

DROP USER
 command, 406-407
 privilege, 187

dropping
 indexes, 441, 700
 roles, 211

rollback segments,
 645, 700

tables, 441, 552, 699

tablespaces, 700

user accounts, 406-407

dummy user accounts, 198-199
 testing privileges, 219

E

e-mail (DBA continuing education), 636

edit command, 89

education/training (DBAs), 630-638, 726
 books, 634
 Internet, 635-637
 Oracle user groups,
 637-638
 test instances, 635
 vendor training programs,
 632-634

efficiency measuring (database speed), 576-578

Engineering Work Group Database Administrators (job description), 10

entity integrity, 596

environmental variables (shell scripts), 371

environments, 56
 client-server architectures,
 271-272
 host-based architectures,
 270
 installing databases, 264
 operating system environ-
 ments, 98-99
 third-party products,
 66-69

error log files, 106

error messages

rollback segments,
645-647

see also troubleshooting

evaluating products, 72

host-based architecture

hardware

components, 269

software, 269

quotes, 269

events, auditing data-bases, 480-481, 483-491**examples**

client-server architectures,
678-679

control file (SQL*Loader),
717

Export utility, 721-722

Import utility, 722-723

exclusive locks, 650, 651**EXECUTE ANY PROCEDURE privilege, 182****EXECUTE privilege, 192**

procedures, 198

executing

shell scripts, 370-373

optimizing queries,
619-622

expanding databases

database administration
schemes, 348

troubleshooting database
speed, 581-583

experimentation (optimizing queries), 626**Export utility, 63, 78,**

90-91, 227, 238-239,

703, 721

example, 721-722

selecting backup

schemes, 241

extents, 156

compressing, 555-559

creating rollback seg-
ments, 647-648

fragmentation, 418-421,
435, 442-443

utilization report (trouble-
shooting tablespaces),
548-551

F**file transfer protocol (ftp) sites, DBA continuing education, 636****files**

alert log files, 702

archive log files, 105, 124,
131-132, 702

*advantages / disadvan-
tages, 228-229*

archivers, 148

*copying redo log files,
129-131*

*database administra-
tion schemes, 346*

*selecting backup
schemes, 240-241*

audit files, 125

cold backups, 232

configuring databases,
database administration
schemes, 340-341,
346-348

control files, 105, 124, 133

*database administra-
tion schemes, 344-345*

*example control file
(SQL*Loader), 717*

data files, 104-105, 124,
127-129

*adding (troubleshooting
tablespaces), 553-555*

*database administra-
tion schemes, 341-344*
*purging old data
files, 559*

*tablespace maintenance,
410-422*

initialization files, 105,
125, 133-135

*adjusting parameters,
304*

locating on disk, 125-127

log files, 125, 135-136

*monitoring script
results, 379*

*troubleshooting data-
bases, 536-538*

*troubleshooting in-
stance / application
crashes, 566-569*

offlineredo log files, 699

operating system file

backups, 226, 231-232

README file, 255, 278,
284-285

table of contents, 284

*upgrading software,
314-315*

redo log files, 105, 124,
129-131, 703

*database administra-
tion schemes, 345-346*

*log writing processes,
147-148*

*modifying active online
file, 699*

*tuning databases,
511-512*

SGA definition file, 106

software files (Oracle), 124

storing (disk space), 287

trace files, 125,
135-136, 704

first normal form, 599
FIRST_ROWS hint, 622
fixed column formats
 (SQL*Loader), compared
 to delimited formats, 716
FORCE ANY TRANSACTION
 privilege, 184
FORCE TRANSACTION
 privilege, 178
foreign keys, 596
formatting
 columns, 707-708
 column command, 368
 *SQL*Plus commands*,
 89
 SQL*Plus output, 706-708
fourth-generation lan-
guage (4GL) tools, 5
fragmentation, 418-421,
435, 442-443
 troubleshooting
 tablespaces, 555-559
 utilization report (monitor-
 ing databases), 464-465
free list (tuning data-
bases), 513
freeing locks, 653
ftp (file transfer protocol)
 sites, DBA continuing
 education, 636
FULL hint, 622
full table scans, 619
full-time database admin-
istrators (job descrip-
tion), 18-23
functions, 664
 SQL*DBA functions, 80-81
future developments, 685
 databases, 52
 Oracle, 6, 632
 Oracle 8, 691

 planning database admin-
 istration schemes,
 336-338

future planning (database
expansion), 518-527
 budget presentations,
 523-527
 calculating system re-
 quirements, 522-523
 historical trends, 518-520
 users' future require-
 ments, 520-522

G

Gantt charts, 356
gateways, 6, 676
get command, 89
goals (optimizing
queries), 615
GRANT ANY PRIVILEGE
 privilege, 182
GRANT ANY ROLE privi-
 lege, 183
grant command,
206-209, 401
granting privileges, 212,
400-402
 privilege sets, 212-219
grants.sql script, 733
group accounts, 398
groups (auditing short-
cuts), 488-489
GUI (graphical user
interface) tools, 5
guides (Installation and
Configuration), 255,
281-283
 appendixes, 283
 installation section, 282
 new software installations
 section, 283
 overview section, 282

 post-installation
 section, 283
 pre-installation
 section, 282
 requirements section, 282
 restrictions section, 282
 upgrading software, 312

H

health of databases,
monitoring, 102, 448-454,
458-460
 configuration report,
 474-477
 DBA job description, 21,
 27, 32
 future planning, 518-527
 scheduling reports,
 459-460
 scripts/reports, 460-477
 security reports, 470-474
 third-party tools, 477-478
 tuning report, 465-470,
 497-498
 utilization report, 462-465
hierarchical storage
 (tuning databases), 503
hints (optimizing queries),
615, 622-623
historical trends (plan-
ning database expan-
sion), 518-520
history of
 computerized data storage,
 44-46
 database administrators
 (DBAs), 7-8, 14-15
 database management
 systems, 45-48
 Oracle, 50-51, 631
 relational databases, 48-50
 system privileges, 172-174

host-based architectures, 58-59, 70, 268

- advantages/disadvantages, 269
- configurations
 - software*, 269
 - VAX system*, 271
- evaluating environment, 270
- implementing Oracle, 70
- installations, troubleshooting, 270
- see also client-based architectures

I

I/O (input/output) operations, tuning databases, 505, 509-510

implementing architectures, 69-73

Import utility, 63, 78,

90-91, 703, 720-723

- command line mode, compared to interactive mode, 720
- example, 722-723
- parameters, 720-721

importing data, 77

INDEX hint, 622

INDEX privilege, 193

- tables, 197

INDEX_ASC hint, 623

INDEX_DESC hint, 623

indexes, 158, 162-164, 596, 601-604, 619

- compressing, 558
- creating, 163, 432-433, 603-604, 697
 - implied indexes*, 603
- deleting, 441, 700

fragmentation, 435,

442-443

locking tables, 652

maintenance, 424-443

calculating sizes,

429-430

creating / designing

indexes, 424-434

monitoring tables /

indexes, 434-436

physical database

structure, 426-431

troubleshooting,

436-441

modifying, 698-699

optimizing queries, 624

primary keys, 602

privileges, 194

sizing, 609

utilization report (monitoring databases), 464

init.ora file, 133

initial_extent tablespace parameter, 157

initialization files, 105,

125, 133-135

adjusting parameters, 304

INSERT ANY TABLE

privilege, 184

INSERT privilege, 193

tables, 197

views, 198

Installation and Configuration Guide, 255, 281-283

appendixes, 283

installation section, 282

new software installations section, 283

overview section, 282

post-installation

section, 283

pre-installation

section, 282

requirements section, 282

restrictions section, 282

upgrading software, 312

installation/configuration

planning (DBA job

description), 20

Installer, 299-301

bugs, 301

creating databases,

304-305

installation, 301

troubleshooting,

302-303

loading, 300-301

terminal emulation,

300-301

upgrading software,

323-324

installing

databases, 257

business needs considerations, 279-281

calculating data

space, 288

evaluating environ-

ments, 264

evaluating products,

266-267

layout recommended by

Oracle, 285-287

memory planning, 289

process planning,

289-290

purchasing components,

262-267

table spaces, 280

tables, 280

troubleshooting,

265-266

user requirements,

280-281

host-based architectures,
troubleshooting, 270

Installer, 301
 troubleshooting,
 302-303

Oracle
 file locations, 125-127
 system requirements, 70

software
 application turnover,
 297
 backups, 298-299
 business needs considerations, 279-281
 CD ROM, 301-303
 configuration, 298
 databases, creating,
 304-305
 environments, evaluating, 264
 Installation and Configuration Guide,
 281-283
 instances, creating, 298
 loading software from tape, 297
 manual creation of databases, 307
 operating system preparations, 297
 Oracle Installer,
 299-301
 products, evaluating,
 266-267
 purchasing components,
 262-267
 README file, 284-285
 tasks, 296-297
 troubleshooting,
 265-266, 305-307
 user requirements,
 280-281

UNIX, 303-304

instances, 702
 auditing databases,
 480-494
 creating (software installation), 298
 definition, 99
 importing data, 77
 monitoring, 458-460
 internal monitoring, 77
 mounting databases, 154
 shutting down, 77,
 231, 246
 starting, 77, 247
 system monitor, 142-143
 tablespace maintenance,
 410-422
 test instances, 635
 troubleshooting, 532-545,
 564-573
 adjusting applications,
 571
 checklist, 586-589
 classifying problem causes, 536-539,
 564-566
 common problems,
 543-544
 expanding Oracle resources, 570
 log files, 536-538,
 566-569
 operating system conflicts, 569-570
 prioritizing problems,
 533-536
 service level agreements,
 544-545
 technical support,
 539-543, 571-573
 unknown problems,
 541-542

upgrading software, 316

interactive mode (Import utility), compared to command line mode, 720

interfaces (non-Oracle applications), 6

internal monitoring of instances, 77

internal white papers, 30

Internet
 DBA continuing education,
 635-637
 technical support, 543

J–K

job submission utilities (database administration schemes), 373-377

killing user sessions (SQL*DBA), 713

Korn shell scripts, 366, 371-373

L

Large UNIX System Database Administrators (job description), 10

latch contention (tuning databases), 511

layout
 installing databases,
 285-287
 Personal Oracle7, 287

LCK (lock writers), 148-149

LGWR (logwriters), 147-148

library cache, 115
 miss rate (tuning databases), 507

lifecyclesof**databases, 253**

installation, 253, 255-257

operations, 253

planning

*installations, 253,**255-256**upgrades, 253, 257-259*

product selection, 253-254

upgrade, 253, 259-260

**linkingupgradedsoft-
ware, 324-325****listeners (SQL*Net), 150****load tables****(SQL*Loader), 716****loading**

data (SQL*Loader), 63, 78,

91-92, 716-717

*badfile optional param-
eter, 717**example control file, 717**fixed column formats,**compared to delimited**formats, 716**load tables, 716**log file optional param-
eter, 717**population scripts, 716*

Installer, 300-301

software

*data space, 288**memory planning, 289**software installation,
297**upgraded, 324-325***Local directory, 126****LOCKANYTABLE privi-
lege, 184****lock writers, 148-149****locks, 650-654**

automatic locks, 652

exclusive locks, 650-651

freeing, 653

manual locks, 652

monitor locks option

(SQL*DBA), 653-654

read-only locking, 651

release time, 653-654

RS locks, 651

RX locks, 652

S locks, 652

shared locks, 650

SRX (share row exclusive)

locks, 652

unique indexes, 652

X locks, 652

**log file optional param-
eter (SQL*Loader), 717****log files, 102, 106, 125,
135-136**

alert log files, 702

archive log files, 105, 124,

131-132, 702

*advantages / disadvan-
tages, 228-229**archivers, 148**copying redo log files,
129-131**database administra-
tion schemes, 346**selecting backup**schemes, 240-241*

error log files, 106

monitoring script

results, 379

offline redo log files, 699

redo log files, 105, 124,

129-131, 703

*database administra-
tion schemes, 345-346**log writing processes,
147-148**modifying active online
file, 699**tuning databases,
511-512*

troubleshooting

*databases, 536-538**instance / application**crashes, 566-569***log writing processes, 138,
140-141, 147-148**

archivers, 148

recoverers, 148

**logical data structures, 37,
154-170, 597**

clusters, 158, 168

compared to physical,

154, 426

databases, 154

indexes, 158, 162-164

sequences, 158, 169

stored procedures, 158,

167-168

synonyms, 158, 166-167

tables, 158, 159-162

tablespaces, 155-157

views, 158, 164-166

login.sql script, 733**logon IDs (user account
maintenance), 396-399****long data type, 161****long raw data type, 161****long-term maintenance
schedules (database
administration schemes),
360-361****lp (lineprinter)****command, 379****M****mailx command, 379****maintenance schedules
(database administration
schemes), 352-362**

daily schedules, 359-360

long-term schedules,

360-361

scheduling conflicts,
357-359
user processing schedules,
355-357
MANAGE TABLESPACE
privilege, 187
manual locks, 652
manually creating data-
bases, 307-309
market share of Oracle
databases, 51-52
Massively Parallel Proces-
sors, 61
max_extents tablespace
parameter, 157
memory, 100, 102-103,
110-121
application memory areas,
119-120
caching tables, 114
cursors (shared pool), 116
data dictionary cache, 116
database buffer cache,
113-114
 data writing processes,
 144-146
multi-threaded servers,
116-117, 118
PGA (Program Global
Area), 111, 117-119
planning installation
procedures, 289
real memory, 111
redo log buffer, 114
SGA (System Global
Area), 111, 112-117
 message storage, 116
 shared pool, 115-116
 shared SQL, 115
shared memory, 111
sort areas, 120
swapping, 111

upgrading software, 316
user requirements for
installation, 280
utilization (tuning data-
bases), 500, 502, 505
virtual memory, 111
menu mode
(SQL*DBA), 712
merge-join algorithm, 620
message storage (SGA
memory), 116
middleware, 677
min_extents tablespace
parameter, 157
mirrored disks, 228, 247
monitorfileio
(SQL*DBA), 713
monitorlocks (SQL*DBA),
653-654, 713
monitor session
(SQL*DBA), 713
monitoring
audit trails, 492-494
databases, 102, 451-454,
458-460
 configuration report,
 474-477
 DBA job description, 21,
 27, 32
 future planning,
 518-527
 scheduling reports,
 459-460
 scripts / reports, 460-477
 security reports,
 470-474
 third-party tools,
 477-478
 tuning report, 465-470,
 497-498
 utilization report,
 462-465

objects (DBA daily routine
overview), 386-387
privileges, 177, 187
processes, 138, 141-142
 process monitors, 144
 system monitors,
 142-143
script results (database
administration schemes),
378-379
tables/indexes, 434-436
tablespaces, 416-418
mounting databases, 154
multi-threaded servers,
116-118, 139, 149-150, 631
parallel processing,
657, 659
processes, 289
tuning databases,
508-509, 511
multimedia databases, 52
multiple database writer
processes, 289
multiprocessing, 656

N

naming conventions,
604-605
columns (tables), 160
database administration
schemes, 332
directories, 287
disks, 287
tables, 159
nested loops (optimizing
queries), 620
Network Manager, 63-64
networks, 676
capacity, 71-72
products, 6
SQL*Net listeners, 150

newsgroups(DBA continuing education), 636

next_extent tablespace parameter, 157

noaudit command, 490

normalized databases, 425, 597-599, 702
 database administration
 schemes, 333-334

number data type, 161

O

object auditing options, 486-488

Object Manager, 65, 86

object privileges, 176, 192-204

 granting, 207

 privilege sets, 199-203

object-by-object backups, 226-227

 Export utility, 238-239

object-oriented

 databases, 52

 programming, 666

ODBC Administrator, 84

OFA(Optimal Flexible Architecture), 125-127, 285-287, 340-341

offline

 redo log files, 699

 rollback segments, 699

 tablespaces, 157

online redo log files, 105, 124, 129-131, 703

 database administration
 schemes, 345-346

 log writing processes,
 147-148

 modifying active online
 file, 699

 offline, 699

 tuning databases, 511-512

operating systems

 backups, 298-299

 conflicts, troubleshooting

 instance/application

 crashes, 569-570

 environments, 98-99

 file backups, 226, 231-232

 logon IDs (user account

 maintenance), 396-399

 software installation, 297

 tuning databases, 504-506

Optimal Flexible Architecture (OFA), 125-127, 285-287, 340-341

OPTIMAL parameter (rollback segments), 644-645

optimizing queries, 612-627

 application design

 options, 616

 cost-based optimizers, 614,
 617-618

 execution plans, 619-622

 experimentation, 626

 goals, 615

 hints, 615, 622-623

 indexes, 624

 rule-based optimizers, 614,
 616-617

 trace utility, 624-625

Oracle

 application development
 tools, 61-63

 background processes, 100

 database monitoring, 102

 definition, 56-57, 682-683

 future developments,
 6, 632

 history, 50-51, 631

 Installer, 299-301

 installing, 257

business needs considerations, 279-281

data space, 288

developing, 290-292

environments, evaluating, 264

file locations, 125-127

Installation and Configuration Guide, 281-283

layout, 285-287

memory area

planning, 289

process planning, 289-290

products, evaluating, 266-267

purchasing components, 262-267

README file, 284-285

reviewers, 292

troubleshooting, 265-266

user requirements, 280-281

 market share, 51-52

 memory processes, 100,
 102-103

 relational database
 management system
 (RDBMS), 4-5

 shutting down instances,
 231, 246

 software files, 124

 starting instances, 247

 system requirements, 70

 third-party products,
 66-69

troubleshooting, 532-545
checklist, 586-589
instance/application
crashes, 564-573
tablespaces, 548-561
technical support,
 571-573

user server processes, 101
 utilities, 63-65
 Windows environment,
 151

Oracle 6 privileges, 177

Oracle 7.2, 690-691

Oracle 8, 691

Oracle base (file architecture), 126

Oracle Installer, 63, 299-301

bugs, 301
 creating databases,
 304-305
 installation, 301
troubleshooting,
 302-303
 loading, 300-301
 terminalemulation,
 300-301
 upgrading software,
 323-324

Oracle Network Manager, 63-64

Oracle Server Manager, 713, 81-82

Oracle Terminal, 63

Oracle user groups, 637-638

Oracle Workgroups Server, 688-690

Oracle*Forms, 62

Oracle*Reports, 62

oracle.cro script, 733

ORACLE_HOME directory, 285

admin tree, 286

orastat script, 733

orastat.2 script, 734

ORDERED hint, 623

P

packages, 665, 670-671

page length commands, 89

parallel processing, 146, 631, 656-662

asynchronous database
 writers, 660
 distributed databases,
 compared to parallel
 servers, 658-659
 multi-threaded
 servers, 659
 parallel queries, 150-151,
 290, 658, 661
 parallel recovery, 661-662

parallel

queries, 150-151, 290,
 658, 661
 recovery, 658, 661-662
 servers
compared to distributed
databases, 658-659
lock writers, 148-149

parameters

Import utility, 720-721
 monitoring databases, 459
 pct_increase tablespace
 parameter, 157
 tuning databases, 496-498,
 506-513

part-time database administrators (job description), 28-32

partial backups

recovery of data, 234-235
 selecting backup schemes,
 240-241

Password Manager, 64, 84

passwords

authenticating, 397
 database security, 34
 roles (granting
 privileges), 211
 script files, 378
 shell scripts (calling
 SQL*Plus), 706
 user account maintenance,
 396-399, 402

patches, 302

pause commands, 89

PC browser tools, 62-63

pccomp script, 734

pct_increase tablespace

parameter, 157

performance, 35-36, 450-451

auditing databases, 491
 organizing data files,
 341-344
 standards of database
 performance, 449-450
see also tuning databases

Personal Oracle 7, 64-65, 82-88, 151

backups, 232-233
 layout, 287

PGA (Program Global Area), 103, 111, 117-119

physical data structures, 426-431, 597

compared to logical,
 154, 426

PL/SQL

stored procedures, 167-168
 programming language,
 279

scripts, 366
database administration schemes, 369-370

planning

calculating disk space
 requirements, 410-416
 database administration
 schemes, 332-333,
 336-338
 database expansion,
 518-527
budget presentations,
 523-527
calculating system requirements, 522-523
historical trends,
 518-520
users' future requirements, 520-522
 for future (DBA job
 description), 21
 installations/configura-
 tions (DBA job descrip-
 tion), 20

PMON(process monitors), 144

population scripts

(SQL*Loader), 716

precision (numbers), 160

presentations (planning database expansion), 523-527

primary keys, 428, 596, 602

prioritizing problems (troubleshooting), 533-536

private rollback segments, 643

private synonyms, 166

privilege sets, 188-189, 212-219

"any" privilege sets,
 189-190

object privileges, 199-203
 scripts, 220-221
 viewing, 403-406

privileges

admin option, 208
 ALTER, 192
sequences, 198
tables, 197
 ALTER ANY CLUSTER,
 181
 ALTER ANY INDEX, 181
 ALTER ANY PROCEDURE,
 182
 ALTER ANY ROLE, 182
 ALTER ANY SEQUENCE,
 183
 ALTER ANY SNAPSHOT,
 183
 ALTER ANY TABLE, 183
 ALTER ANY TRIGGER,
 184
 ALTER DATABASE, 186
 ALTER PROFILE, 186
 ALTER RESOURCE
 COST, 186
 ALTER ROLLBACK
 SEGMENT, 186
 ALTER SESSION, 178
 ALTER SYSTEM, 186
 ALTER TABLESPACE,
 187
 ALTER USER, 187
 ANALYZE ANY, 181
"any" privileges, 177,
 180-185
privilege sets, 189-190
 AUDIT ANY, 181
 AUDIT SYSTEM, 187
 BACKUP ANY TABLE,
 184
 BECOME USER, 187
 column privileges, 198

COMMENT ANY TABLE,
 184
 CREATE ANY CLUSTER,
 181
 CREATE ANY INDEX,
 181
 CREATE ANY PROCEDURE,
 182
 CREATE ANY SEQUENCE,
 183
 CREATE ANY SNAPSHOT,
 183
 CREATE ANY SYNONYM,
 183
 CREATE ANY TABLE,
 183
 CREATE ANY TRIGGER,
 184
 CREATE ANY VIEW, 185
 CREATE CLUSTER, 179
 CREATE DATABASE
 LINK, 179
 CREATE PROCEDURE,
 179
 CREATE PROFILE, 186
 CREATE PUBLIC DATA-
 BASE LINK, 186
 CREATE PUBLIC SYNONYM,
 179
 CREATE ROLE, 186
 CREATE ROLLBACK
 SEGMENT, 186
 CREATE SEQUENCE,
 179
 CREATE SESSION, 178
 CREATE SNAPSHOT, 180
 CREATE SYNONYM, 180
 CREATE TABLE, 180
 CREATE TABLESPACE,
 187
 CREATE TRIGGER, 180

- CREATE VIEW, 180
- database maintenance
 - privileges, 177, 185-187
- database privileges, 194
- DELETE, 192
 - tables*, 197
 - views*, 197
- DELETE ANY TABLE, 184
- developer privileges, 177, 178-180
- DROP ANY CLUSTER, 181
- DROP ANY INDEX, 181
- DROP ANY PROCEDURE, 182
- DROP ANY ROLE, 183
- DROP ANY SEQUENCE, 183
- DROP ANY SNAPSHOT, 183
- DROP ANY SYNONYM, 183
- DROP ANY TABLE, 184
- DROP ANY TRIGGER, 184
- DROP ANY VIEW, 185
- DROP PROFILE, 186
- DROP PUBLIC DATABASE LINK, 186
- DROP PUBLIC SYNONYM, 179
- DROP ROLLBACK SEGMENT, 186
- DROP USER, 187
- dummy user accounts, 198-199
- EXECUTE, 192, 198
- EXECUTE ANY PROCEDURE, 182
- FORCE ANY TRANSACTION, 184
- FORCE TRANSACTION, 178
- GRANT ANY PRIVILEGE, 182
- GRANT ANY ROLE, 183
- granting, 206-209, 212, 400-402
- INDEX, 193-194
 - tables*, 197
- INSERT, 193
 - tables*, 197
 - views*, 198
- INSERT ANY TABLE, 184
- LOCK ANY TABLE, 184
- MANAGE TABLESPACE, 187
- monitoring databases, 470-474
- monitoring privileges, 177, 187
- object privileges, 176, 192-204
- Oracle 6 privilege sets, 177
- privilege sets, 188-189, 212-219
 - "any" privilege sets*, 189-190
 - object privileges*, 199-203
 - scripts*, 220-221
 - viewing*, 403-406
- procedure privileges, 194, 198
- public synonym privileges, 195
- querying views, 207-208
- REFERENCES
 - privilege, 193
 - tables*, 197
- revoke command, 207
- roles, 174, 209-212
- SELECT, 193
 - sequences*, 198
 - tables*, 197
 - views*, 198
- SELECT ANY SEQUENCE, 183
- SELECT ANY TABLE, 184
- sequence privileges, 169, 194, 198
- stored procedures, 168
- synonym privileges, 166-167, 195
- system privileges, 172-190
 - auditing databases*, 484
 - history*, 172-174
- table privileges, 161, 194, 196-197
- tablespace privileges, 194
- tablespace quota
 - option, 209
- UNLIMITED TABLESPACE, 180
- UPDATE, 193
 - tables*, 197
 - views*, 198
- UPDATE ANY TABLE, 184
- user account maintenance, 395-396
 - changing privileges*, 402-406
- user privileges, 177-178
- view privileges, 194, 197-198
- prob_chk.txt script, 734**
- procedural option, 279**
- procedures**
 - privileges, 194, 198
 - see also* stored procedures
- process monitors, 141, 144**

process.sql script, 734**processes**

- background processes, 99-100, 138-151
 - data writing processes*, 138-140, 144-146
 - dedicated servers*, 149
 - lock writers*, 148-149
 - log writing processes*, 138-141, 147-148
 - monitoring processes*, 138, 141-142
 - multi-threaded servers*, 149-150
 - parallel query processes*, 150-151
 - process monitors*, 144
 - SQL*Net listeners*, 150
 - system monitors*, 142-143
 - user service processes*, 138-139
- data writing processes*, 138-140, 144-146
- log writing processes*, 147-148
- Oracle 7 processes, 151
- parallel query processes, 150-151
- planning installation
 - procedures, 289-290
- recoverer process, 290
- user service processes, 138-139

processing, 146, 631,**656-662**

- architectures, 60-61
- parallel
 - asynchronous database writers*, 660

- distributed databases*, compared to *parallel servers*, 658-659
- multi-threaded servers*, 659
- parallel queries*, 150-151, 290, 658, 661
- parallel recovery*, 661-662
- user processing schedules, 355-357

Product directory, 126**product reviews**

- Oracle 7.2, 690-691
- Oracle Workgroups Server, 688-690

product stacks,**defined, 272****production environment, 24**

- transferring applications, 28

Program Global Area

- (PGA), 103, 111, 117-119

prompt command, 368, 708**ps (process status) command, 378****public rollback**

- segments, 643

public synonyms, 166

- privileges, 195

purchasing products,

- vendors, 274-276

purging old data files, 559

Q**queries, 77, 155**

- indexes, 162-164, 601-604
- optimizing, 612-627
 - application design options*, 616

- cost-based optimizers*, 614, 617-618
- execution plans*, 619-622
- experimentation*, 626
- goals*, 615
- hints*, 615, 622-623
- indexes*, 624
- rule-based optimizers*, 614-617
- trace utility*, 624-625

- parallel queries, 661

- parallel query processes, 150-151

- summary calculation
 - tables, compared to
 - views, 606

- tuning (database speed), 580-581

quotas (tablespaces), troubleshooting, 550-551

R**raw data**

- files, 128
- types, 161

RDBMS (relational database management system), 4-5, 594-597

- data independence*, 597
- designing tables for*
 - decision support, 600-601
- entity integrity*, 596
- foreign keys*, 596
- history*, 48-50
- indexes*, 601-604
- logical data*
 - structures, 597
- normalizing databases*, 597-599

- physical data
 - structures, 597
- primary keys, 596
- referential integrity, 596
- read command, 371**
- read-only**
 - locking, 651
 - tablespaces, 129, 343
- README file, 255, 278, 284-285**
 - table of contents, 284
 - upgrading software, 312-315
- real memory, 111**
- RECO (recoverers), 148**
- recording transactions (logfiles), 102**
- recoverer process, 290**
- recoverers, 141, 148**
- Recovery Manager, 65, 86-87**
- recovery of data, 77, 703**
 - (DBA job description), 21, 27, 31, 33-34
 - cold backups, 233-236
 - disaster recovery, 702
 - parallel recovery, 661-662
 - partial backups, 234-235
 - testing backups, 225, 235-236
- recursive cursors, 116**
- redo log**
 - buffer, 114
 - files, 105, 124, 129-131, 703
 - database administration schemes, 345-346*
 - log writing processes, 147-148*
 - modifying active online file, 699*

- offline, 699*
- tuning databases, 511-512*
- REFERENCES**
- privilege, 193**
 - tables, 197
- referential integrity, 596**
 - database administration schemes, 334
- relational database management system (RDBMS), 4-5, 594-597**
 - data independence, 597
 - designing tables for
 - decision support, 600-601
 - entity integrity, 596
 - foreign keys, 596
 - history, 48-50
 - indexes, 601-604
 - logical data
 - structures, 597
 - normalizing databases, 597-599
 - physical data
 - structures, 597
 - primary keys, 596
 - referential integrity, 596
- relational database model, 594-597**
- reliability of**
 - databases, 450
- RESOURCE (Oracle 6 privilege set), 177**
- resource contention (tuning databases), 506-513**
- responsibilities of DBAs, 8-10, 14-42, 683-684**
 - backup/recovery, 21, 27, 31, 33-34
 - computer system support, 18

- continuing education, 22
 - books, 634*
 - Internet, 635-637*
 - Oracle user groups, 637-638*
 - test instances, 635*
 - vendor training programs, 632-634*
- data accuracy, 37
- database performance, 35-36
- developer as database administrator, 23-28
- developer support, 17-18
- expertise level
 - expected, 17
- full-time database administrators, 18-23
- logical data structures, 37
- monitoring database
 - health, 21, 27, 32
- part-time database administrators, 28-32
- percentage of time available, 17
- planning for future, 21
- reviewing new products, 21-22, 28
- security of database, 33-35
- size of database system, 17
- specialization requirements, 18
- support staff relationships, 37-39, 40-42
- troubleshooting, 21, 27, 32
 - (DBA job description), 21, 27, 32*
 - checklist, 586-589*
 - classifying problem causes, 536-539*
 - common problems, 543-544*

- database administration schemes*, 332
 - database speed*, 576-584
 - DBA daily routine overview*, 388-390
 - installation of architectures*, 270-272
 - installation (Oracle)*, 262-267
 - installation procedures*, 282-285
 - instances*, 564-573
 - prioritizing problems*, 533-536
 - rollback segments*, 551, 645-647
 - service level agreements*, 544-545
 - system tablespace*, 551
 - table/index maintenance*, 436-441
 - tablespaces*, 548-561
 - technical support*, 539-543
 - unknown problems*, 541-542
 - upgrading software*, 325-326
 - updating data, 32
 - user administration, 21, 27, 32
 - user support, 21, 28, 39-42
 - reviewers (testing development of installation procedures)**, 292
 - reviewing new products (DBA job description)**, 21-22, 28
 - revoke**
 - command, 207, 402
 - privileges, user account maintenance, 402-406
 - roles, 174, 209-212**
 - creating, 210, 401
 - default roles, 210-211
 - dropping, 211
 - granting privileges, 400-402
 - passwords, 211
 - privilege sets, 212-219
 - user account maintenance, 395-396
 - changing privileges*, 402-406
 - rollback segment tablespaces, 344, 703**
 - rollback segments, 642-648, 703**
 - creating, 643, 647-648
 - deleting, 700
 - dropping, 645
 - error messages, 645-647
 - offline, 699
 - OPTIMAL parameter, 644-645
 - private rollback segments, 643
 - public rollback segments, 643
 - storage space, 644-645
 - SYS user, 643
 - troubleshooting, 551
 - tuning databases, 510
 - root.sh shell script, 303**
 - rotating backup schemes, 242-245**
 - routers, 676**
 - routine maintenance**
 - daily schedules, 359-360
 - database administration schemes, 352-362
 - long-term schedules, 360-361
 - scheduling conflicts, 357-359
 - user processing schedules, 355-357
 - row cache, 116**
 - row IDs, 159**
 - rowid data type, 161**
 - RS locks, 651**
 - rule-based optimizers, 614-617**
 - RX locks, 652**
-
- ## S
-
- S locks, 652**
 - save command, 89**
 - saving DDL scripts, 560**
 - scale (numbers), 160**
 - scheduling**
 - database monitoring reports, 459-460
 - DBA daily routine overview, 384-391
 - monitoring objects*, 386-387
 - scheduled events*, 386
 - troubleshooting*, 388-390
 - user support*, 387-388
 - routine maintenance
 - daily schedules*, 359-360
 - database administration schemes*, 352-362
 - long-term schedules*, 360-361
 - scheduling conflicts*, 357-359
 - user processing schedules*, 355-357
 - scripts, 290**
 - adduser script, 732
 - alert script, 732

- backup schemes, 245-247
- bench script, 732
- CATEXP.SQL script, 721
- cold backups, 231-236
 - files needed*, 232
 - recovery of data*, 233-236
 - selecting backup schemes*, 240
- config.sql script, 732
- creating tables/indexes, 433-434
- database administration
 - schemes, 364-380
 - automated job submission utilities*, 373-377
 - automating DBA tasks*, 365-367
 - monitoring script results*, 378-379
 - PL/SQL scripts*, 369-370
 - shell scripts*, 370-373
 - SQL scripts*, 367-369
 - testing scripts*, 377-378
- database monitoring
 - configuration report*, 474-477
 - reports*, 460-477
 - security reports*, 470-474
 - tuning report*, 465-470
 - utilization report*, 462-465
- dbch_bat script, 732-733
- dbcheck script, 733
- dbcheck.2 script, 733
- disk contents, 732-735
- file, 89, 93
- grants.sql script, 733
- login.sql script, 733
- oracle.cro script, 733

- orastat script, 733
- orastat.2 script, 734
- password files, 378
- pccomp script, 734
- privilege sets, 220-221
- prob_chk.txt script, 734
- process.sql script, 734
- root.sh shell script, 303
- saving DDL scripts, 560
- sec_ck.sql script, 734
- sec_role.txt script, 734
- security, password
 - files, 378
- shell scripts (calling SQL*Plus), 706
 - database administration schemes*, 370-373
 - environmental variables*, 371
- sizer.sql script, 734
- SQL*Plus, 708-709
- starter script, 734
- t_cklist.txt script, 735
- tab_size.xls script, 735
- tune_ck.sql script, 735
- util_ck.sql script, 735
- warm backups, 226, 236-238
 - database administration schemes*, 343
 - ending*, 237
 - selecting backup schemes*, 240-241
 - starting*, 236
- searching databases, 162-164**
- sec_ck.sql script, 734**
- sec_role.txt script, 734**
- security**
 - auditing databases, 482-483, 491-492

- data corruption, 35
- database administration
 - schemes, 330-334
- DBA job description, 33-35
- disaster recovery, 35
- granting privileges, 212
- health of databases, 451
- object privileges, 176, 192-204
- passwords, 34
- privileges, granting, 206-209
- reports, 453, 470-474
- roles (granting privileges), 209-212
- scripts, password files, 378
- stored procedures, 668
- system privileges, 172-190
- user account maintenance, 394-396
- SELECT ANY SEQUENCE privilege, 183**
- SELECT ANY TABLE privilege, 184**
- SELECT privilege, 193**
 - sequences, 198
 - tables, 197
 - views, 198
- selecting backup schemes, 239-242**
- sequences, 158, 169**
 - privileges, 169, 194, 198
- Server Manager, 81-82, 713**
- service level agreements (troubleshooting databases), 544-545**
- Session Manager, 64, 83**
- set pagesize command, 89, 706-707**

set pause command,
89, 707

set role command, 211

SGA (System Global Area),
103, 111-117

definition file, 106

shared locks, 650

shared memory, 111

**shared pool (SGA
memory),**
115-116

**shared SQL (SGA
memory),** 115

**shell scripts (calling
SQL*Plus),** 706

database administration

schemes, 370-373

environmental

variables, 371

**shortcuts, auditing data-
bases,** 488-489

show sga command, 117

shutting down instances,
71, 231, 246

sizer.sqlscript, 734

sizing

indexes, 609

tables, 606-609

Small UNIX System

**Database Administrators
(job description),** 10

SMON (system monitor),
142-143

**SMP (Symmetrical Multi-
Processors),** 61, 656

software

configuring host-based
architectures, 269

files (Oracle), 124

installation

application turnover,
297

backups, 298-299

*business needs consider-
ations,* 279-281

CD ROM, 301-303

configuration, 298

creating databases,
304-305

creating databases,
manually, 307

creating instances, 298

*evaluating environ-
ments,* 264

evaluating products,
266-267

loading software, 297

operating system

preparations, 297

Oracle Installer,
299-301

problems, 265-266,
305-307

purchasing components,
262-267

tasks, 296-297

user requirements,
280-281

**Installation and Configu-
ration Guide,** 281-283

appendixes, 283

installation section, 282

*new software installa-
tions section,* 283

overview section, 282

*post-installation
section,* 283

*pre-installation
section,* 282

*requirements
section,* 282

restrictions section, 282

upgrading software, 312

linking, upgraded,
324-325

loading

data space, 288

memory planning, 289

software installation,
297

upgraded, 324-325

memory areas, 119-120

purchasing products,
vendors, 274-276

README file, 255, 278,
284-285

table of contents, 284

upgrading software,
312-315

storing in directories,
285-287, 314

admin tree, 286

ORACLEHOME

directory, 285

sub tree, 286

storing in databases,
664-666, 671

functions, 664

*object-oriented program-
ming,* 666

packages, 665-666,
670-671

stored procedures, 664,
668-670

triggers, 665, 667

upgrading, 259-260,
315-317

backing out, 317-318

backups, 324

bug fixes, 315

*changes needed in the
database,* 315

- Installation and Configuration Guide*, 312
- Installer*, 323-324
- instances, cleaning*, 316
- issues*, 257-258
- linking new software*, 324-325
- loading new software*, 324-325
- memory*, 316
- new features*, 315-316
- planning*, 258-259
- README file*, 312-315
- storing new software*, 314
- support*, 318
- testing*, 316
- troubleshooting*, 325-326
- software code, memory processes**, 102
- Software Engineering Institute (Carnegie Mellon University)**, 450
- sort areas (memory)**, 103, 120
- sort memory contention, tuning databases**, 512-513
- sorting data (database administration schemes)**, 334
- specialization requirements (of DBAs)**, 18
- speed of databases, troubleshooting**, 576-584
 - application tuning, 580-581
 - efficiency measuring, 576-578
 - tuning databases, 579-580
 - user expectations, 578-579
- spool command**, 89, 368, 707-708
- SQL (Structured Query Language)**
 - accessing tables, 161
 - privilege set scripts, 220-221
 - queries, 155
 - application design options*, 616
 - cost-based optimizers*, 614, 617-618
 - execution plans*, 619-622
 - experimentation*, 626
 - goals*, 615
 - hints*, 615, 622-623
 - indexes*, 162-164, 601-604, 624
 - optimizing*, 612-627
 - rule-based optimizers*, 614-617
 - trace utility*, 624-625
 - parallel queries*, 150-151, 661
 - summary calculation tables, compared to views*, 606
 - tuning (database speed)*, 580-581
- scripts, 366
 - database administration schemes*, 367-369
- shared SQL (SGA memory), 115
- SQL*DBA**, 63, 78-81, 703, 712-713
 - command line mode, 712
 - compared to SQL*Plus, 709
 - executing scripts, 370-371
 - functions, 80-81
 - killing user sessions, 713
 - menu mode, 712
 - monitor locks option, 653-654
 - monitoring capabilities, 713
- SQL*Loader**, 63, 78, 91-92, 716-717
 - badfile optional parameter, 717
 - example control file, 717
 - fixed column formats, 716
 - load tables, 716
 - log file optional parameter, 717
 - population scripts, 716
- SQL*Net**, 64
 - listeners, 150
 - listeners process, 290
 - log files, 135
 - multi-threaded servers, 149-150
 - user account maintenance, 398
- SQL*Plus**, 61-62, 78, 88-90, 703, 706-709
 - calling, 706
 - compared to SQL*DBA, 709
 - executing shell scripts, 370-373
 - formatting columns, 89
 - output formatting, 706-708
 - reports, 367-369
 - scripts, 708-709

sqlplus command, 706**SRX (share row exclusive)****locks, 652****standards (database)**

efficiency, 576-578

performance, 449-450

starter script, 734**starting**

database audits, 481-482

instances, 247

warm backups, 236

instances, 77

statement auditing**options, auditing databases, 484-486****stopping**

instances, 77

warm backups, 237

storage requirements,**rollback segments, 644-645****stored procedures, 158,****167-168, 424, 664, 668-670**

privileges, 168

see also procedures**storing**

data, 99, 104-106, 154-170

*clusters, 158, 168**databases, 154**disk space, 287**history, 44-46**indexes, 158, 162-164**rollback segments, 644-645**sequences, 158, 169**stored procedures, 158, 167-168**synonyms, 158, 166-167**tables, 158-162**tablespaces, 155-157**views, 158, 164-166*

software, in directories, 285-287, 314

*admin tree, 286**ORACLE_HOME**directory, 285**sub tree, 286*

software in databases,

664-666, 671

*functions, 664**object-oriented programming, 666**packages, 665-666, 670-671**stored procedures, 664, 668-670**triggers, 665, 667***striping disks (tuning databases), 503****Structured Query Language (SQL)**

accessing tables, 161

privilege set scripts, 220-221

queries, 155

*application design options, 616**cost-based optimizers, 614, 617-618**execution plans, 619-622**experimentation, 626**goals, 615**hints, 615, 622-623**indexes, 162-164, 601-604, 624**optimizing, 612-627**rule-based optimizers, 614-617**trace utility, 624-625**parallel queries, 150-151, 661**summary calculation**tables, compared to**views, 606**tuning (database speed), 580-581*

shared SQL (SGA memory), 115

subtrees, storing software, 286**summary calculation tables, compared to views, 606****summary tables, 430****support staff relationships (DBA job description), 37-39, 40-42****support**

tools

*distributed option, 279**procedural option, 279*

upgrading software, 318

swapping memory, 111**symmetric multiprocessor (SMP), 656****Symmetrical Multi-Processors (SMP), 61****synonyms, 158, 166-167, 424, 596**

private synonyms, 166

privileges, 166-167, 195

public synonyms, 166

SYS user, rollback segments, 643**sys user ID, 398-399****system administrators, 19-20****system configuration analyses, 728-730****System Global Area (SGA), 103, 111-117****system integrators, 73**

system monitors, 141, 142-143

system privileges, 172-190

auditing databases, 484
granting, 207
history, 172-174

system requirements

calculating for database
expansion, 522-523
troubleshooting database
speed, 581-583
installing Oracle, 70

system tablespace, 703

storing software
objects, 665
troubleshooting, 551

system user ID, 398-399

T

t_cklist.txt script, 735

tab_size.xls script, 735

tables, 158-162, 596

backups, Export
utility, 227, 238-239
caching in memory, 114
clusters, 158, 168
compressing, 557
creating, 159-160, 431,
431-432, 696-697
data types, 161
de-fragmenting, 438-439
deleting, 699
designing for decision
support, 600-601
dropping, 441
troubleshooting
tablespaces, 551-552
fragmentation, 435,
442-443
history of relational
databases, 48-49

indexes, 162-164

installing databases, 280

maintenance, 424-443

creating / designing
tables, 424-434
monitoring tables /
indexes, 434-436

physical database
structure, 426-431
troubleshooting,
436-441

modifying, 440-441, 698

naming, 159

normalized tables, data-
base administration
schemes, 333-334

privileges, 161, 194-197
sizing, 606-609

SQL (Structured Query
Language)

accessing tables, 161
privilege set scripts,
220-221
queries, 155
shared SQL (SGA
memory), 115

summary calculation
tables, compared to
views, 606

summary tables, 430

synonyms, 166-167

views, 164-166

tablespaces, 155-157

creating, 696

data files, 127-128

default storage param-
eters, 156-157

deleting, 700

extents, 156

compressing, 555-559

creating rollback

segments, 647-648

fragmentation, 418-421,
435, 442-443

utilization report
(troubleshooting
tablespaces), 548-551

fragmentation, 419-421

granting privileges, 209

installing databases, 280

listing, 155-157

maintenance, 410-422

calculating disk space
requirements, 410-416
monitoring tablespaces,
416-418

troubleshooting,
418-421

modifying, 698

new users, creating 400

offline tablespaces, 157

privileges, 194

quotas, troubleshooting,
550-551

read-only tablespaces,
129, 343

reorganization, 421

rollback segment

tablespaces, 344, 703

system tablespaces, 703

storing software

objects, 665

troubleshooting, 551

temporary tablespaces,
343, 703

tools tablespace, 704

troubleshooting, 548-561
adding data files,
553-555

allocating space from
other tablespaces, 559

- compressing extents*, 555-559
- de-fragmentation*, 555-559
- dropping tables*, 551-552
- purging old data files*, 559
- saving DDL scripts*, 560
- utilization report*, 548-551
- utilization report (monitoring databases), 462-463
- warm backups (database administration schemes), 343
- tape drives, user requirements for installation**, 281
- TAR directory**, 126
- technical support**, 539-543, 571-573, 726
- temporarily disabling user accounts**, 407
- temporary tablespaces**, 343, 703
- Terminal**, 63
- terminal emulation, Installer**, 300-301
- test instances**, 635
- testing**
 - backups, 235-236
 - recovery of data, 225
 - scripts (database administration schemes), 377-378
 - software, upgrading, 316
- third-party products**, 66-69, 93
 - monitoring databases, 477-478
- threads**, 656
- three-tier client-server environments**, 674
- tips/suggestions summary**, 684
- tools tablespace**, 704
- trace files**, 106, 125, 135-136, 704
 - troubleshooting instance/application crashes, 566-569
- trace utility (optimizing queries)**, 624-625
- training**, 630-638, 726
 - books, 634
 - Internet, 635-637
 - Oracle user groups, 637-638
 - test instances, 635
 - vendor training programs, 632-634
- transactions, locks, release time**, 653-654
- transferring data (SQL*Loader)**, 63, 78, 91-92, 716-717
 - badfile optional parameter, 717
 - example control file, 717
 - fixed column formats, compared to delimited formats, 716
 - load tables, 716
 - log file optional parameter, 717
 - population scripts, 716
- triggers**, 665-667
- troubleshooting**, 532-545
 - (DBA job description), 21, 27, 32
 - checklist, 586-589
 - classifying problem causes, 536-539
 - common problems, 543-544
 - database administration schemes, 332
 - database speed, 576-584
 - application tuning*, 580-581
 - efficiency measuring*, 576-578
 - system requirements (database expansion)*, 581-583
 - tuning databases*, 579-580
 - user expectations*, 578-579
 - DBA daily routine overview, 388-390
 - installation of architectures
 - host-based architectures*, 270
 - client-server architectures*, 272
 - installation (Oracle)
 - evaluating environment*, 264-267
 - evaluating products*, 266-267
 - problems*, 265-266
 - purchasing components*, 262-267
 - installation procedures
 - Installation and Configuration Guide*, 281-283
 - README file*, 284-285
 - instances, 564-573
 - adjusting applications*, 571
 - classifying problem causes*, 564-566

- expanding Oracle resources, 570*
- log files, 566-569*
- operating system conflicts, 569-570*
- technical support, 571-573*
- prioritizing problems, 533-536
- rollback segments, 551, 645-647
- service level agreements, 544-545
- system tablespace, 551
- table/index maintenance, 436-441
- tablespaces, 548-561
 - adding data files, 553-555*
 - allocating space from other tablespaces, 559*
 - compressing extents, 555-559*
 - de-fragmentation, 555-559*
 - dropping tables, 551-552*
 - maintenance, 418-421*
 - purging old data files, 559*
 - quotas, 550-551*
 - saving DDL scripts, 560*
 - utilization report, 548-551*
- technical support, 539-543
- unknown problems, 541-542
- upgrading software, 325-326
- trusted version (Oracle), 70**
- tune_ck.sqlscript, 735**

- tuning applications (database speed), 580-581**
- tuning databases, 448, 455, 496-515, 579-580**
 - application processing requirements, 501-502
 - balanced disk loading, 502
 - buffer cache hit ratio, 509
 - checklist, 513-514
 - computing power, 499-500
 - CPU utilization, 504-505
 - data dictionary cache miss rate, 507-508
 - disk space, 500-501
 - disk striping, 503
 - free list, 513
 - hierarchical storage, 503
 - I/O operations, 505, 509-510
 - latch contention, 511
 - library cache miss rate, 507
 - memory utilization, 500, 502, 505
 - multi-threaded server processes, 511
 - multi-threaded server session memory, 508-509
 - operating system indicators, 504-506
 - redo log buffer space contention, 511-512
 - reports, 452-453, 465-470
 - resource contention, 506-513
 - rollback segments, 510
 - sort memory contention, 512-513
 - see also performance*
- turnkey solutions, 6**

U

- uninterruptable power supplies (UPS), 270**
- unique indexes, 428**
 - locking tables, 652
- UNIX, installation, 303-304**
- unknown problems, troubleshooting, 541-542**
- UNLIMITED TABLESPACE privilege, 180**
- UPDATE ANY TABLE privilege, 184**
- UPDATE privilege, 193**
 - tables, 197
 - views, 198
- updating data (DBA job description), 32**
- upgrading**
 - databases, 325
 - difference from installations, 312
 - software, 259-260, 315-317
 - backing out, 317-318*
 - backups, 324*
 - bug fixes, 315*
 - changes needed in the database, 315*
 - Installation and Configuration Guide, 312*
 - Installer, 323-324*
 - instances, cleaning, 316*
 - issues, 257-258*
 - linking new software, 324-325*
 - loading new software, 324-325*
 - memory, 316*
 - new features, 315-316*

- planning*, 258-259
- README file*, 312-315
- storing new software*, 314
- support*, 318
- testing*, 316
- troubleshooting*, 325-326

UPS (uninterruptable power supplies), 270

USE_MERGE hint, 623

USE_NL hint, 623

user account maintenance, 394-408

- adding new users, 399-402
- changing privileges, 402-406
- deleting user accounts, 406-407
- granting privileges, 400-402
- group accounts, 398
- operating system logon IDs, 396-399
- passwords, 402
- security schemes, 394-396
- SQL*Net, 398
- temporarily disabling users, 407

users

- administration (DBA job description), 21, 28-32
- expectations (database speed), 578-579
- future requirements
 - planning database expansion, 520-522
- groups (DBA continuing education), 637-638
- privileges, 177-178
- processing schedules (database administration schemes), 355-357

- requirements (database administration schemes), 335-336
- server processes (Oracle), 101
- service processes, 138-139
- sessions, killing user sessions (SQL*DBA), 713
- support

- (DBA job description)*, 21, 28, 39-42
- DBA daily routine overview*, 387-388

- User Manager, 65, 86

util_ckpt.sql script, 735

utilities, 63-65

- at utility (UNIX), 374
- Backup Manager, 64, 83-84
- cron utility (UNIX), 374-377
- Database Expander, 64, 85-86
- Database Manager, 64, 83
- Export, 63, 78, 90-91, 227, 238-239, 721
 - example*, 721-722
 - selecting backup schemes*, 241
- Import, 63, 78, 90-91, 703, 720-723
 - command line mode, compared to interactive mode*, 720
 - example*, 722-723
 - parameters*, 720-721
- job submission utilities (database administration schemes), 373-377
- Object Manager, 65, 86
- ODBC Administrator, 84
- Oracle Installer, 63

- Oracle Network Manager, 63-64
- Oracle Terminal, 63
- Password Manager, 64, 84
- Recovery Manager, 65, 86-87
- Server Manager, 81-82
- Session Manager, 64, 83
- SQL*DBA, 63, 78, 78-81, 703, 712-713
 - command line mode*, 712
 - compared to SQL*Plus*, 709
 - executing scripts*, 370-371
 - functions*, 80-81
 - killing user sessions*, 713
 - menu mode*, 712
 - monitor locks option*, 653-654
 - monitoring capabilities*, 713
- SQL*Loader, 63, 78, 91-92, 716-717
 - badfile optional parameter*, 717
 - example control file*, 717
 - fixed column formats*, 716
 - load tables*, 716
 - log file optional parameter*, 717
 - population scripts*, 716
- SQL*Net, 64
 - listeners*, 150
 - listeners process*, 290
 - log files*, 135
 - multi-threaded servers*, 149-150
 - user account maintenance*, 398

SQL*Plus, 61-62, 78,
88-90, 703, 706-709
 calling, 706
 *compared to SQL*DBA*,
 709
 executing shell scripts,
 370-373
 formatting columns, 89
 output formatting,
 706-708
 reports, 367-369
 scripts, 708-709
trace utility (optimizing
 queries), 624-625
User Manager, 65, 86
utilization report
 monitoring databases, 452,
 462-465
 troubleshooting
 tablespaces, 548-551

V

varchar2 data type, 161
variables, environmental
variables, shell scripts,
371

**VAX system, configura-
tion**, 271
**vendor training pro-
grams**, 632-634
vendors
 purchasing products,
 274-276
 technical support, 30-31
views, 158, 164-166,
424, 596
 auditing databases, 482,
 490-491
 compared to summary
 calculation tables, 606
 creating, 165
 designing tables for
 decision support, 600-601
 monitoring tablespaces,
 416-418
 privilege sets, 403-406
 privileges, 194, 197-198
 querying about privileges,
 207-208
 synonyms, 166-167
virtual memory, 111

W

warm backups, 226,
236-238
 database administration
 schemes, 343
 ending, 237
 selecting backup schemes,
 240-241
 starting, 236
**whenever clause (auditing
databases)**, 489
white papers, 30
**Windows, Oracle 7 pro-
cesses**, 151
Workgroups Server, 688-
690
WorldWideWeb (WWW)
 DBA continuing education,
 636-637
 technical support, 543
writing to data files, 101

X-Y-Z

Xlocks, 652